

COURSE OBJECTIVES:

- To learn the basic web concepts and Internet protocols.
- To understand CGI Concepts & CGI Programming.
- To familiarize with Scripting Languages.
- To study DHTML, XML, SERVELETS AND JSP.

LEARNING OUTCOMES:

- Demonstrate an understanding of the components of a computer information networked system, including application and software, communication protocols, and networking hardware and software. □
- Create, install and update sophisticated web sites. Install and manage server software and other server side tools. □
- Demonstrate critical thinking in the understanding, evaluation and application of technology solutions to a variety of real-life situations. □
- Articulate ethical and professional standards as they apply to the use of the computer systems and computer based data. □

UNIT I Introduction**(9)**

Internet Principles – Basic Web Concepts – Client/Server model – retrieving data from Internet – HTML and Scripting Languages – Standard Generalized Mark –up languages – Next Generation – Internet – Protocols and Applications.

UNIT-II Common gateway interface programming**(9)**

CGI Concepts – HTML tags Emulation – Server – Browser Communication – E-mail generation – CGI client Side applets – CGI server applets – authorization and security.

UNIT III Scripting languages**(9)**

HTML – forms – frames – tables – web page design- XML - JavaScript introduction – control structures – functions – arrays – objects – simple web applications

UNIT IV Dynamic HTML**(9)**

Dynamic HTML – introduction – cascading style sheets – object model and collections – event model – filters and transition – data binding – data control – ActiveX control – handling

of multimedia data

UNIT V Servlets and JSP

(9)

JSP Technology Introduction-JSP and Servlets- Running JSP Applications Basic JSP- JavaBeans Classes and JSP-Tag Libraries and Files- Support for the Model-View- Controller Paradigm- Case Study-Related Technologies.

Total Hours:
45

TEXT BOOKS:

1. Deitel H.M. and Deitel P.J., “Internet and World Wide Web How to program”, Pearson International, 2012, 4th Edition. (Ch-1,4,5,6,12,14,26,27)
2. Uttam K.Roy, “Web Technologies”, Oxford University Press, 2011.

REFERENCES:

1. Gopalan N.P. and Akilandeswari J., “Web Technology”, Prentice Hall of India, 2011.(Ch- 1 to 11)
2. Paul Dietel and Harvey Deitel,”Java How to Program”, Prentice Hall of India, 8th Edition.(Ch-29),2012
3. Mahesh P. Matha, “Core Java A Comprehensive study”, Prentice Hall of India, 2011.
4. Thomno A. Powell,” The Complete Reference HTML and XHTML”, Tata McGraw Hill, 2008.

WEBSITES:

1. www.wileyindia.com/web-technologies-html-javascript-php-java-jsp-xml
2. www.comptechdoc.org/



KARPAGAM ACADEMY OF HIGHER EDUCATION

Faculty of Engineering

Department of Computer Science and Engineering

Lecture Plan

Subject Name: INTERNET AND WEB TECHNOLOGY

Subject Code: 16BECS702

S.No	Topic Name	No.of Periods	Supporting Materials	Teaching Aids
UNIT- I INTRODUCTION				
1	Introduction	1	R[1]-1	BB
2	Internet principles	1	R[1]-1	BB
3	Basic web concepts	1	R[1]-5	PPT
4	Client/Server model	1	R[1]-6	PPT
5	Retrieving data from internet	1	R[1]-6	PPT
6	HTML and Scripting Languages	1	R[1]-95	PPT
7	HTML and Scripting Languages	1	R[1]-95	PPT
8	Standard Generalized Mark –up languages	1	R[1]-68	BB
9	Next Generation	1	Web	PPT
10	Internet Protocols and Applications.	1	R[1]-12	BB
11	Tutorial : Scripting Programming	1	Web	BB
Total		11		
UNIT- II COMMON GATEWAY INTERFACE PROGRAMMING				
12	CGI Concepts	1	R[1]-200	PPT
13	HTML tags Emulation	1	web	PPT
14	Server – Browser Communication	1	R[1] 201	BB
15	E-mail generation	1	R[1]214	PPT
16	E-mail generation	1	R[1]214	PPT
17	CGI client Side applets	1	R[1]218	PPT
18	CGI server applets	1	R[1]218	PPT
19	CGI server applets	1	R[1]218	PPT
20	Authorization and security	1	R[1]221	BB
21	Tutorial : HTML	1	R[1]221	PPT
Total		10		
UNIT- III SCRIPTING LANGUAGES				
22	HTML – forms	1	web	PPT

23	Frames ,Tables	1	web	PPT
24	Web page design	1	web	PPT
25	XML	1	T[1]-488	BB
26	JavaScript introduction	1	T[1]-193	PPT
27	Control structures	1	T[1]-266	BB
28	Functions	1	T[1]-305	PPT
29	Arrays, Objects	1	T[1]-343	BB
30	Simple Web Applications	1	web	PPT
31	Tutorial: Web page Design	1	web	PPT
Total		10		
UNIT- IV DYNAMIC HTML				
32	Dynamic HTML	1	R[1]-139	PPT
33	Dynamic HTML	1	R[1]-139	PPT
34	Cascading style sheets	1	T[1]-140	PPT
35	Object model and Event model	1	R[1]-152	BB
36	Filters and Transition	1	R[1]-159	PPT
37	Data binding	1	R[1]-162	BB
38	Data control	1	R[1]-163	PPT
39	ActiveX control	1	R[1]-133	PPT
40	Handling of multimedia data	1	web	PPT
41	Tutorial : ActiveX control	1	R[1]-133	BB
Total		10		
UNIT- V SERVLETS AND JSP				
42	JSP Technology Introduction	1	R[1]-248	PPT
43	JSP and Servlets	1	R[1]-465	BB
44	JSP and Servlets	1	R[1]-465	BB
45	Running JSP Applications	1	R[1]-255	PPT
46	Basic JSP	1	R[1]-248	PPT
47	JavaBeans Classes	1	T[1]-1087	PPT
48	JSP-Tag Libraries and Files	1	T[1]-1087	PPT
49	Support for the Model View- Controller Paradigm	1	T[1]-690	BB
50	Case Study,Related Technologies	1	T[1]-690	PPT
51	Tutorial: Running JSP Applications	1	T[1]-752	BB
52	Discussion on Previous University Question Papers			
Total		10		
Total Hours		52		

TEXT BOOKS

S.N O	Title of the book			Year of publication
----------	-------------------	--	--	------------------------

1	Deitel H.M. and Deitel P.J., “Internet and World Wide Web How to program”, Pearson International, 2012, 4th Edition. (Ch-1,4,5,6,12,14,26,27)			2012
2	Uttam K.Roy, “Web Technologies”, Oxford University Press, 2011.			2011

REFERENCE BOOKS

S.N O	Title of the book			Year of publication
1	Gopalan N.P. and Akilandeswari J., “Web Technology”, Prentice Hall of India, 2011.(Ch- 1 to 11)			2011
2	Paul Dietel and Harvey Deitel,”Java How to Program”, Prentice Hall of India, 8th Edition.(Ch-29),2012			2012
3	Mahesh P. Matha, “Core Java A Comprehensive study”, Prentice Hall of India, 2011.			2011

WEBSITES

1. www.wileyindia.com/web-technologies-html-javascript-php-java-jsp-xml
2. www.comptechdoc.org/independent/web/

UNIT I

INTRODUCTION TO INTERNET

1. INTERNET

The Internet is a worldwide system of interconnected computer networks. The computers and computer networks exchange information using TCP/IP (Transmission Control Protocol/Internet Protocol). The computers are connected via the telecommunications networks, and the Internet can be used for e-mailing, transferring files and accessing information on the World Wide Web.

- In 1969 the precursor of Internet is born: ARPAnet.
- ARPA = Advanced Research Projects Agency sponsored by the American Department of Defense (DOD). Designed to connect military research centers.
- Distributed computer system able to survive a nuclear attack.
- Problem: ARPAnet could connect only networks of the same type.
- In 1970, ARPA starts developing the Transmission Control Protocol / Internet Protocol (TCP/IP), a technology for connecting networks of different types (produced by different companies).
- Other networks appear, such as CSNET and BITNET.

2. SERVICE PROVIDED BY THE INTERNET

a) Electronic Mail (E-mail)

- Electronic mail, commonly known as email or e-mail, 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.
- Email servers accept, forward, deliver and store messages.
- An email message consists of three components, the message envelope, the message header, and the message body. The message header contains control information, including attachment option, reason box, email address and one or more recipient addresses. Message can consist of attachments, graphic or video/audio clips.
- E.g.: - e-mail addresses • Samsung@gmail.com • Apple@outlook.com
- Some popular E-mail services providers are: • 1. Gmail • 2. Hotmail • 3. Yahoo • 4. MSN

b) World wide web

World Wide Web (WWW) Important features of the world wide web (www) are listed below,

- Most important service provided by Internet.
- An internet-based hypermedia initiative for global information sharing.
- Developed in 1989 by Tim Berners-Lee of the European Particle Physics Lab (CERN) in Switzerland.

c) File Transfer Protocol

- File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet.
- FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.

d) Chat Rooms

- Real time typed conversation via computers.
- Chat rooms (the channel or medium).
- Chat clients (program used to connect to a chat server)
- Normally included on a browser
- Freely downloaded from the web
- Some are text only, others support voice & video

e) Mailing list

A mailing list is a collection of names and addresses used by an individual or an organization to send material to multiple recipients. The term is often extended to include the people subscribed to such a list, so the group of subscribers is referred to as "the mailing list", or simply "the list".

- Group of e-mail address given a single name
- When a message is sent to the mailing list everyone on the list receive the message
- To add your name to a mailing list you must subscribe to it, to remove your name you must unsubscribe.

f) Instant Messaging

Instant Messaging Instant messaging (IM) is a type of online chat which offers real-time text transmission over the Internet. A LAN messenger operates in a similar way over a local area network.

- Notifies you when one or more people are online allow exchange of messages and files
- It allows you to join a private chat rooms.
- Real time conversation that takes place on a computer
- Chat room is location on server that permits users to discuss topics of interest
- Some are the text only others support voice and video

g) News groups

A news group is called as forum, an on-line discussion group. On the Internet, there are literally thousands of newsgroups covering every conceivable interest. To view and post messages to a newsgroup, you need a news reader, a program that runs on your computer and connects you to a news server on the Internet.

- Online area in which users conduct written discussion about a particular subject.
- Usenet (collection of all internet news groups).
- News server (computer storing newsgroups messages).
- Newsreader (program used to access newsgroups).

3. INTERNET PRINCIPLES

Basic Internet Principles

- TCP/IP
- UDP
- IP Addresses
- Domain Names
- The Domain Name System (DNS)
- Ports
- Sockets
- URLs

TCP/IP

The Internet is the network that connects computers all over the world. It works according to a set of agreed-upon 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. The IP address of the sender (or source, or origin)
3. The IP address of recipient (or 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. Ensures delivery of data in the correct order
- IP is concerned with *routing*. IP attaches the address of the destination of each packet. IP ensures that packets get to the right place.
 - TCP is the higher-level protocol that uses the lower-level IP.
 - When an application is written, the general principle is to use the highest level protocol that you can, provided that it provides the functionality and performance that is required.

Many applications can be written using TCP/IP. For example, a Web browser can be written in Java using only URLs, without any explicit mention of sockets.

- On each machine an application program makes calls on procedures in the transport layer (normally TCP). In turn the transport layer makes calls on the Internet layer (normally IP). In turn the Internet layer makes calls on the physical layer, which is different depending on the technology of the communication link.
- At the destination machine, information is passed up through the layers to the application program. Each application program acts as if it is communicating directly with the application on another machine. The lower levels of the communication software and hardware are invisible.
- This four-layer model is sufficient for understanding Internet software. But there are other models that use a different number of layers, like the ISO seven-layer model.
- The application layer produces some data, adds a header to it and passes the complete package to the transport layer. The transport layer adds another header and passes the package to the internet layer. The internet layer adds another header and passes it to the physical layer. The application data is enclosed by 4 headers used by the different layers. This process can be thought of as repeatedly putting a letter into an envelope and then addressing the envelope.

UDP

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is **unreliable and connectionless protocol**. So, there is no need to establish connection prior to data transfer.

Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of Internet services, provides assured delivery, reliability and much more but all these services cost us with additional overhead and latency. Here, UDP comes into picture. For the real-time services like computer gaming, voice or video communication, live conferences, we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also saves bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

UDP Header –

UDP header is **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. First 8 Bytes contains all necessary header information and remaining part consists of data. UDP port number fields are each 16 bits long, therefore range for port numbers defined from 0 to 65535, port number 0 is reserved. Port numbers help to distinguish different user requests or process.

1. **Source Port** : Source Port is 2 Byte long field used to identify port number of source.
2. **Destination Port** : It is 2 Byte long field, used to identify the port of destined packet.
3. **Length** : Length is the length of UDP including header and the data. It is 16-bits field.
4. **Checksum** : Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, pseudo header of information from the IP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

IP Addresses

An *IP address* is a unique address for every host computer in the world. It consists of 4 bytes or 32 bits. This is represented in *quad notation* (or *dot notation*) as four 8-bit numbers, each in the range 0 to 255, e.g. 131.123.2.220.

IP addresses are registered so that they stay unique.

You can find the IP address of the local machine under Windows NT by typing the following command at the DOS prompt in a console window:

ipconfig

Under Unix or Linux, this command is:

ifconfig

The IP address 127.0.0.1 is a special address, called the *local loopback address* that denotes the local machine. A message sent to this address will simply return to the sender, without leaving the sender. It is useful for testing purposes.

Domain Names

A *domain name* is the user-friendly equivalent of an IP address. It is used because the numbers in an IP address are hard to remember and use. It is also known as a *host name*.

Example:

cs.stmarys.ca

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 is something like com, org, edu, ca, 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.

The Domain Name System

A program, say a Web browser, that wants to use 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 located in one place, so it is held in a *distributed database*.

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, or 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 generally 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 of peer-to-peer relationships: chat, internet phone, shared whiteboard.

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. A port number does not correspond to any physical connection on the machine, of which there might be just one. Each type of service has, by convention, a standard port number. Thus 80 usually mean 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.

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 at 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.

Streams

Accessing information across the Internet is accomplished using streams. A stream is a serial collection of data. An output stream can be sent to a printer, a display, a serial file, or an Internet connection, for example. Likewise, an input stream can come from a keyboard, a serial file, or from an Internet connection. Thus reading or writing to another program across a network or the Internet is just like reading or writing to a serial file.

URL

A URL (*Uniform Resource Locator*):

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

A URL has this structure:

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

- Things in square brackets indicate that the item can be omitted.
- 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.

The first part of a URL is the particular protocol. Some commonly-used protocols are:

http	The service is the Web. The file is accessed using the HTTP protocol.
ftp	The service is file transfer protocol. The URL locates a file, a directory or an FTP server.
telnet	The service is remote login to a host. No file name is needed.
mailto	The service is e-mail.
news	The URL specifies a usenet newsgroup.
file	This locates a file on the local system. The server part of the URL is omitted.

A pathname (optional) specifies a directory (folder). The pathname is not the complete directory name, but is relative to some directory (folder) designated by the administrator as the

directory in which publicly-accessible files are held. It would be unusual for a server to make available its entire file system to clients.

The file name can either be a data file name or can specify an executable file that produces a valid HTML document as its output. A file name is often omitted. In this case, the server decides which file to use. Many servers send a default file from the directory specified in the path name - for example a file called default.html, index.html or welcome.html.

The **section** part of a URL (optional) specifies a named anchor in an HTML document. Such a place in a document is specified by an HTML entry like:

```
<a name="url"></a>
```

4. BASIC WEB CONCEPTS

Clients, servers, and sites

- On the Web, information is stored at **Web sites**.
- Access to the information at a Web site is managed by a **Web server** for the site.
- Users access the information using a **Web client**, which is also called a **browser**.

How information is stored: HTML

Information on the Web is stored in documents, using a language named **HTML** (HyperText Markup Language). Web clients interpret HTML and 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).

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 link in the document shown in their Web client. Documents and locations within documents are identified by an address that is called a Uniform Resource Locator, or **URL**.

This is an example of a URL:

```
http://www.sybase.com/products
```

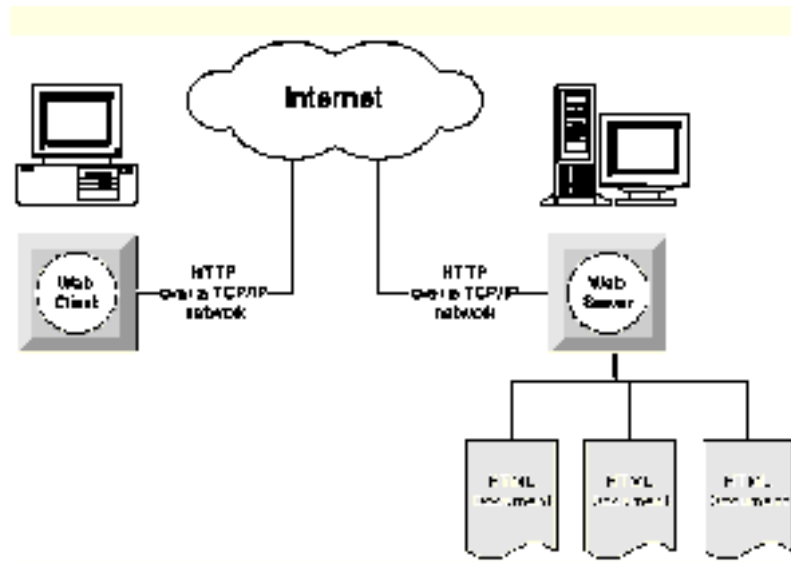


Figure 4.1 *Storing, locating and transmitting information on the web*

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 in 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 across the network to the Web client.

External data in HTML documents

HTML documents can reference external files (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.

Problems with file-based Web sites

In a file-based Web site, each document is a separate file. For large Web sites, this leads to management problems. For example, maintaining current copies of hundreds or thousands of different resources in separate files is difficult enough; maintaining the links between these resources is even more challenging.

Another problem is that many Web sites contain dynamic information - pricing information, for example, or employee information on an organization's intranet. Maintaining such information in HTML files in addition to the database where it resides is a huge task.

For these and other reasons, linking databases to the Web is increasingly the solution of choice for management of large Web sites and management of dynamic content. Database storage of Web information can either replace or complement file storage of Web resources.

5. CLIENT SERVER MODEL

Although an Internet system provides a basic communication service, the protocol software cannot initiate control with, or accept contact from, a remote computer.

Of course, two application involved in a communication **cannot both** wait for a message to arrive. One application must actively initiate interaction while the other application passively waits.

Most network applications use a form of communication known as **the client –server paradigm**. A server application waits passively for contact, while a client application initiates communication actively.

Client Server Model

A client and server networking model is a model in which computers such as servers provide the network services to the other computers such as clients to perform a user based tasks. This model is known as client-server networking model.

The application programs using the client-server model should follow the given below strategies:

- An application program is known as a client program, running on the local machine that requests for a service from an application program known as a server program, running on the remote machine.
- A client program runs only when it requests for a service from the server while the server program runs all time as it does not know when its service is required.
- A server provides a service for many clients not just for a single client. Therefore, we can say that client-server follows the many-to-one relationship. Many clients can use the service of one server.
- Services are required frequently, and many users have a specific client-server application program. For example, the client-server application program allows the user to access the files, send e-mail, and so on. If the services are more customized, then we should have one generic application program that allows the user to access the services available on the remote computer.

Process classification

- **Client process**, the process that requires a service
- **Server process**, the process that provides the required service

Client

A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the service started by the user and terminates when the service is completed.

Server

A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.

A server program is an infinite program means that when it starts, it runs infinitely unless the problem arises. The server waits for the incoming requests from the clients. When the request arrives at the server, then it responds to the request.

A server must guarantee:

- Authentication: client identity verification
- Authorization: verification of the possibility for a client to access to a particular service
- Data security: guarantee that specific data cannot be read and/or modified

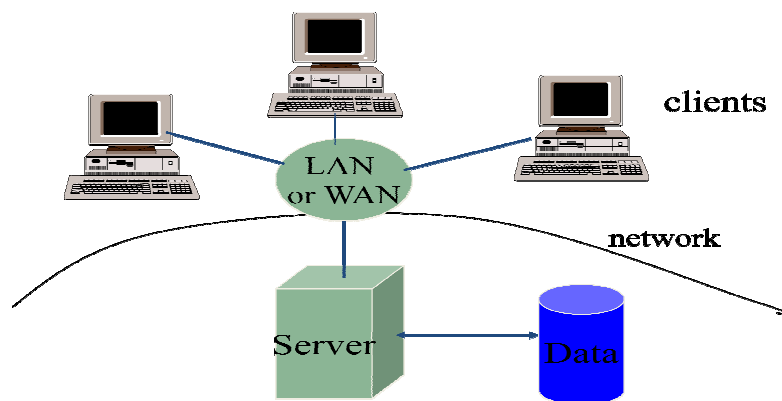


Figure 5.1 Architecture of Client Server Model

Establishing a Connection

- While establishing connection, both client and server will construct a socket.
- A socket is one end of an inter-process communication channel.
- The process to establish a socket on the client side is different than the process to establish a socket on the server side.

Establish a Socket on Client Side

- Create a socket using the `socket()` system call.
- Connect the socket to the address of the server using the `connect()` system call

- Send and receive data. You can use read() and write() system calls. However, there are different ways to send and receive data.

Establish a Socket on Client Side

- Create a socket using the socket() system call.
- Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number of the host machine.
- Listen for the connections using the listen() system call.
- Accept a connection using the accept() system call
- Send and receive data. You can use read() and write() system calls. However, there are different ways to send and receive data.

Address domain and the socket type

- The client and server processes can communicate with each other only if their sockets are in the same domain and of the same type.
- Two widely used address domains are:
 - UNIX domain (in which two processes share a common file system).
 - Internet domain (in which two processes run on any two hosts on the Internet)

UNIX and Internet domain

- The address of a socket in the Unix domain is a character string, an entry in the file system.
- The address of a socket in the Internet domain is the Internet address (IP address) of the host machine.
- Each socket needs a port number (16 bits unsigned integers) on the host.
- The lower numbers are reserved in Unix for standard services (e.g. 21 is used as the port number for the FTP server).
- Generally, port number above 2000 are available.

Types of Socket

- Two widely used socket types are:
 - Stream socket: treats communications as a continuous stream of characters. It uses TCP
 - Datagram socket: reads entire message at one time. It uses UDP.

Characteristics of Client-server architecture

- Client and server machines need different amount of hardware and software resources.
- Client and server machines may belong to different vendors.
- *Horizontal scalability* (increase of the client machines) and *vertical scalability* (migration to a more powerful server or to a multiserver solution)
- A client or server application interacts directly with a transport layer protocol to establish communication and to send or receive information.
- The transport protocol then uses lower layer protocols to send or receive individual messages. Thus, a computer needs a complete stack of protocols to run either a client or a server.

- A single server-class computer can offer multiple services at the same time; a separate server program is needed for each service.
- **Identifying a particular service** TCP uses 16-bit integer values (protocol port numbers) to identify services, and assign a unique port number to each service.
- A client specifies the protocol port number of the desired service when sending a request.

Advantages of Client-server networks:

- **Centralized:** Centralized back-up is possible in client-server networks, i.e., all the data is stored in a server.
- **Security:** These networks are more secure as all the shared resources are centrally administered.
- **Performance:** The use of the dedicated server increases the speed of sharing resources. This increases the performance of the overall system.
- **Scalability:** We can increase the number of clients and servers separately, i.e., the new element can be added, or we can add a new node in a network at any time.

Disadvantages of Client-Server network:

- **Traffic Congestion** is a big problem in Client/Server networks. When a large number of clients send requests to the same server may cause the problem of Traffic congestion.
- It does not have a robustness of a network, i.e., when the server is down, then the client requests cannot be met.
- A client/server network is very decisive. Sometimes, regular computer hardware does not serve a certain number of clients. In such situations, specific hardware is required at the server side to complete the work.
- Sometimes the resources exist in the server but may not exist in the client. For example, if the application is web, then we cannot take the print out directly on printers without taking out the print view window on the web.

6. HTML

HTML stands for Hypertext Markup Language, and it is the most widely used language to write Web Pages.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
- As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

HTML Tags

As told earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within angle braces **<Tag Name>**. Except few tags, most of the tags have their corresponding closing tags. For example, **<html>** has its closing tag **</html>** and **<body>** tag has its closing tag **</body>** tag etc.

Above example of HTML document uses the following tags –

Sr.No	Tag & Description
1	<!DOCTYPE...> This tag defines the document type and HTML version.
2	<html> This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
3	<head> This tag represents the document's header which can keep other HTML tags like <title> , <link> etc.
4	<title> The <title> tag is used inside the <head> tag to mention the document title.
5	<body> This tag represents the document's body which keeps other HTML tags like <h1> , <div> , <p> etc.
6	<h1> This tag represents the heading.
7	<p> This tag represents a paragraph.

HTML Document Structure

A typical HTML document will have the following structure –

```
<HTML>

    <head>

        HTML header related tags

    </head>

    <body>

        Document body related tags

    </body>

</HTML>
```

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration –

```
<!DOCTYPE html>
```

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Paragraph Tag

The **<p>** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening **<p>** and a closing **</p>** tag as shown below in the example –

Line Break Tag

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **
** tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use **
** it is not valid in XHTML.

Centering Content

You can use **<center>** tag to put any content in the center of the page or any table cell.

Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The `<hr>` tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag `<pre>`.

Any text between the opening `<pre>` tag and the closing `</pre>` tag will preserve the formatting of the source document.

HTML Elements

An **HTML element** is defined by a starting tag. If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags –

Start Tag	Content	End Tag
<code><p></code>	This is paragraph content.	<code></p></code>
<code><h1></code>	This is heading content.	<code></h1></code>
<code><div></code>	This is division content.	<code></div></code>
<code>
</code>		

So here `<p>...</p>` is an HTML element, `<h1>...</h1>` is another HTML element. There are some HTML elements which don't need to be closed, such as `<img.../>`, `<hr />` and `
` elements. These are known as **void elements**.

HTML Attributes

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts – a **name** and a **value**

- The **name** is the property you want to set. For example, the paragraph `<p>` element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.

- The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: **left**, **center** and **right**.

Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are –

- Id
- Title
- Class
- Style

The Id Attribute

The **id** attribute of an HTML tag can be used to uniquely identify any element within an HTML page. There are two primary reasons that you might want to use an id attribute on an element –

- If an element carries an id attribute as a unique identifier, it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

The title Attribute

The **title** attribute gives a suggested title for the element.

The class Attribute

The **class** attribute is used to associate an element with a style sheet, and specifies the class of element. You will learn more about the use of the class attribute when you will learn Cascading Style Sheet (CSS). So for now you can avoid it.

The style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

Generic Attributes

Here's a table of some other attributes that are readily usable with many of the HTML tags.

Attribute	Options	Function
align	right, left, center	Horizontally aligns tags

valign	top, middle, bottom	Vertically aligns tags within an HTML element.
bgcolor	numeric, hexadecimal, RGB values	Places a background color behind an element
background	URL	Places a background image behind an element
id	User Defined	Names an element for use with Cascading Style Sheets.
class	User Defined	Classifies an element for use with Cascading Style Sheets.
width	Numeric Value	Specifies the width of tables, images, or table cells.
height	Numeric Value	Specifies the height of tables, images, or table cells.
title	User Defined	"Pop-up" title of the elements.

HTML - Formatting

Anything that appears within `...` element, is displayed in bold.

- **Italic Text**

Anything that appears within `<i>...</i>` element is displayed in italicized

- **Underlined Text**

Anything that appears within `<u>...</u>` element, is displayed with underline

- **Superscript Text**

The content of a `^{...}` element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

- **Subscript Text**

The content of a `_{...}` element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

- **Larger Text**

The content of the `<big>...</big>` element is displayed one font size larger than the rest of the text surrounding it

- **Smaller Text**

The content of the `<small>...</small>` element is displayed one font size smaller than the rest of the text surrounding it

- **Grouping Content**

The `<div>` and `` elements allow you to group together several elements to create sections or subsections of a page.

For example, you might want to put all of the footnotes on a page within a `<div>` element to indicate that all of the elements within that `<div>` element relate to the footnotes. You might then attach a style to this `<div>` element so that they appear using a special set of style rules.

HTML – Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the `<table>` tag in which the `<tr>` tag is used to create table rows and `<td>` tag is used to create data cells. The elements under `<td>` are regular and left aligned by default

- **Table Heading**

Table heading can be defined using `<th>` tag. This tag will be put to replace `<td>` tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element in any row. Headings, which are defined in `<th>` tag are centered and bold by default.

- **Cell padding and Cell spacing Attributes**

There are two attributes called *cell padding* and *cell spacing* which you will use to adjust the white space in your table cells. The cell spacing attribute defines space between table cells, while cell padding represents the distance between cell borders and the content within a cell.

- **Colspan and Rowspan Attributes**

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

HTML - Text Links

- **Linking Documents**

A link is specified using HTML tag `<a>`. This tag is called **anchor tag** and anything between the opening `<a>` tag and the closing `` tag becomes part of the link and a user can click that part to reach to the linked document. Following is the simple syntax to use `<a>` tag.

- **The target Attribute**

We have used **target** attribute in our previous example. This attribute is used to specify the location where linked document is opened. Following are the possible options –

Sr.No	Option & Description
1	_blank Opens the linked document in a new window or tab.
2	_self Opens the linked document in the same frame.
3	_parent Opens the linked document in the parent frame.
4	_top Opens the linked document in the full body of the window.
5	targetframe Opens the linked document in a named <i>targetframe</i> .

Insert Image

You can insert any image in your web page by using **** tag. Following is the simple syntax to use this tag.

```
<img src = "Image URL"...attributes-list/>
```

The **** tag is an empty tag, which means that, it can contain only list of attributes and it has no closing tag.

7. SCRIPTING LANGUAGES

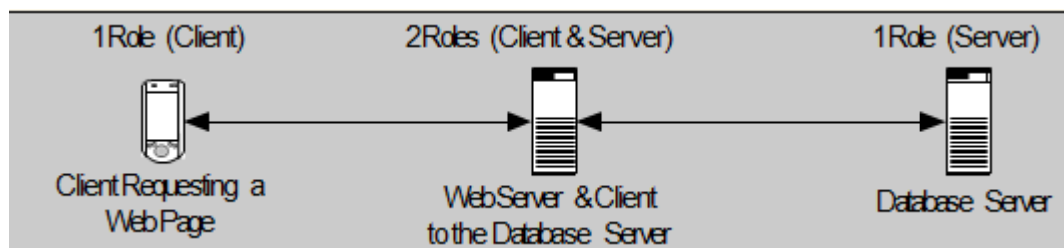
A scripting language is a programming language designed for integrating and communicating with other programming languages. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++.

There are many scripting languages some of them are discussed below:

- **Bash:** It is a scripting language to work in the Linux interface. It is a lot easier to use bash to create scripts than other programming languages. It describes the tools to use and code in the command line and creates useful reusable scripts and conserves documentation for other people to work with.
- **Node js:** It is a framework to write network applications using **JavaScript**. Corporate users of Node.js include IBM, LinkedIn, Microsoft, Netflix, PayPal, and Yahoo for real-time web applications.
- **Ruby:** There are a lot of reasons to learn Ruby programming language. Ruby's flexibility has allowed developers to create innovative software. It is a scripting language which is great for web development.
- **Python:** It is easy, free and open source. It supports procedure-oriented programming and object-oriented programming. Python is an interpreted language with dynamic semantics and a huge line of code are scripted and is currently the most hyped language among developers.
- **Perl:** A scripting language with innovative features to make it different and popular. Found on all windows and Linux servers. It helps in text manipulation tasks. High traffic websites that use Perl extensively include priceline.com, IMDB.
- **PHP:** PHP is a recursive acronym for "PHP: Hypertext Preprocessor". PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- **Javascript:** JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

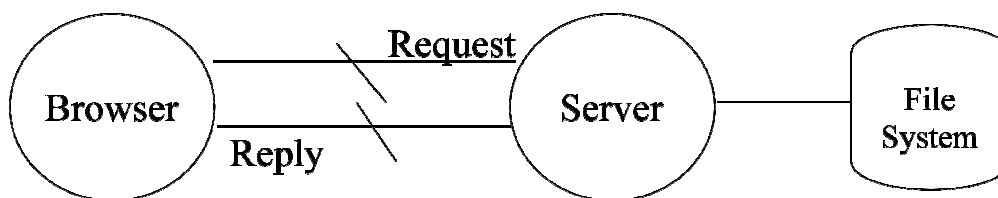
client-side and server-side

- Any machine can play the role of either a client or a server
 - You could even have a machine being both
- Some languages, e.g. Javascript, are said to be *client-side*.
 - Run on the user's browser/web client
- Other languages, e.g. PHP, are said to be *server-side*.
 - Run on the server that is delivering content to the user



Servers

- Two important examples are:
 - Web Servers
 - Database Servers
 - Some of these machines may be powerful computers dedicated to the task, i.e. Database server
- A web server is a machine holding and delivering HTML pages that can be accessed by remote (client) machines over the Internet.
- **Static Web Model :** Client sends a request to the server for a web page. The server looks up the web page using part of the URL you have sent it, and then returns the HTML page which your browser subsequently displays on your machine.

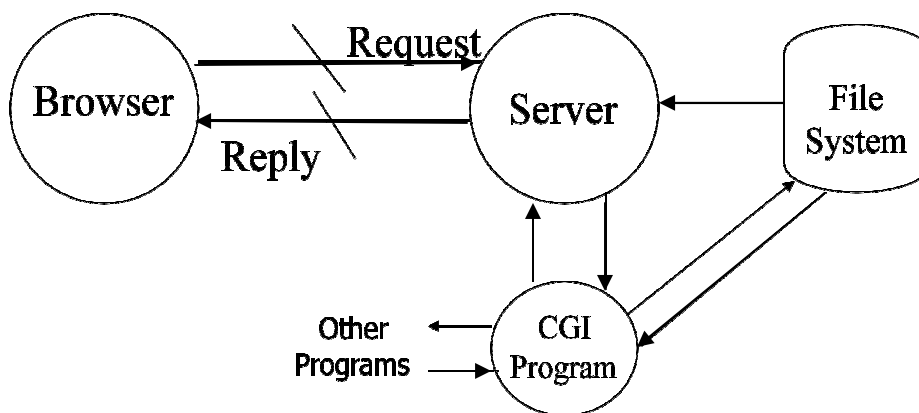


- **Dynamic model:** Client sends a request to the server and it dynamically determines the HTML that is to be returned.
- The dynamics of the reply is achieved through extending the *web server* with a program (script) that does some data processing and creates HTML output based on the data you

sent (e.g. contents of a form). The process of generating the HTML response is performed *server-side*.

SERVER SIDE SCRIPTING

- Dynamic web model
 - One approach is the Common Gateway Interface (CGI) where we have a separate program that can be executed.
 - An alternative is to have extra code in the HTML that can be executed on the *server* to determine the HTML that is to be returned.
 - That is how PHP works.



CLIENT SIDE SCRIPTING

- The other (complementary) approach is to do the work on the client machine.
 - Again we have extra code in the HTML, but now it is executed by the user's browser (i.e. *client-side*).
 - Most common client side script is Java script.
 - An example of its use is when a web page has a form. We can use Java script to validate the input data client-side before it is sent to a server.
 - If we do the validation on the client, this reduces the work that the server has to do and reduces the time taken to respond to the user.
- HTML5 essentially includes Java script elements to enhance its power.
- Java script can also be used to create dynamic web page content. For example:
 - We could change the content based on the fact that you visited the web page before.
 - Time of day.
 - JavaScript popup menus.

Simple Java script program

```
<html>

<head>

<title>A First Program in JavaScript</title>

</head>

<body>

<h1>Dynamic generation</h1>

<script language = "JavaScript">

    document.writeln("<p>Welcome to JavaScript programming</p>");

</script>

</body>

</html>
```

Advantages of scripting languages

- **Easy learning:** The user can learn to code in scripting languages quickly, not much knowledge of web technology is required.
- **Fast editing:** It is highly efficient with the limited number of data structures and variables to use.
- **Interactivity:** It helps in adding visualization interfaces and combinations in web pages. Modern web pages demand the use of scripting languages. To create enhanced web pages, fascinated visual description which includes background and foreground colors and so on.
- **Functionality:** There are different libraries which are part of different scripting languages. They help in creating new applications in web browsers and are different from normal programming languages.

Application of Scripting Languages

Scripting languages are used in many areas:

- Scripting languages are used in web applications. It is used in server side as well as client side. Server side scripting languages are: JavaScript, PHP, Perl etc. and client side scripting languages are: JavaScript, AJAX, jQuery etc.
- Scripting languages are used in system administration. For example: Shell, Perl, Python scripts etc.
- It is used in Games application and Multimedia.
- It is used to create plug-in and extensions for existing applications.

8. Standard Generalized Mark-up languages (SGML)

SGML is a language for recording and storing document information—a computer language that nonetheless can also be read and understood by humans. You use SGML to write rules that a group of related documents should follow when they store your desired added value. The general process of figuring out the rules is called *modeling*, and when you're done modeling and expressing the model in SGML form, your set of rules serves as the “language” that these documents use to “speak” to computers. In SGML terminology, such a group of documents is described by the term *document type*, and a rule set for a document type is called a *document type definition*, or *DTD*.

For example, newspapers might be a document type, and your SGML rules for adding information to the newspapers' electronic files would be a newspaper DTD. Such information might include, for instance, the date each feature article was assigned to be written and the name of the news organization from which each photo was obtained.

SGML, Document Types, and Documents

DTDs form the foundation for every SGML-based document production system by:

- Rigorously recording and enforcing your requirements for document intelligence and structure
- Controlling the text editors that insert and keep track of the added intelligence and structure, allowing some authoring functions to be automated
- Controlling the systems that manage whole and partial documents
- Providing information about the documents to the software that formats them, indexes them for retrieval, and otherwise processes them

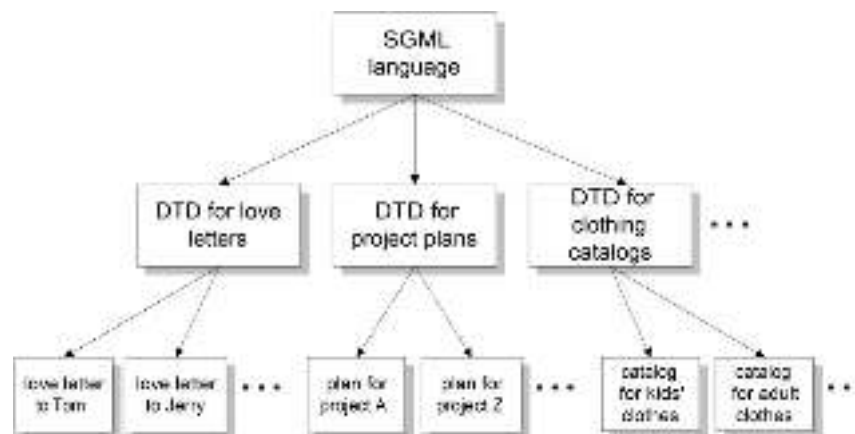


Figure 8.1. SGML, DTDs, and Document Instances

The SGML language can be used to express any number of DTDs. Likewise, each DTD can be used as the basis for many documents of the same general type, with each document being an *instance*—an example—of the type described by its DTD. Any set of similar documents can be considered a document type to be modeled using SGML DTD rules: love letters, project plans, product catalogs, and so on.

SGML Markup Strengths

To summarize, SGML markup is unique in that it combines several design strengths:

- It is declarative, which helps document producers “write once, use many”—putting the same document data to multiple uses, such as delivery of documents in a variety of online and paper formats and interchange with others who wish to use the documents in different ways.
- It is generic across systems and has a nonproprietary design, which helps make documents vendor and platform independent and “future-proof”—protecting them against changes in computer hardware and software.
- It is contextual, which heightens the quality and completeness of processing by allowing documents to be structurally validated and by enabling logical collections of data to be manipulated intelligently.

SGML Constructs

The grammar of every SGML markup language has four basic “parts of speech”: elements, attributes, entities, and comments. In the following sections we'll use a hypothetical set of DTD rules for a very simple “recipe” document type, along with two real recipes, to illustrate these parts of speech.

Elements

SGML and Other Markup Systems described the notion of nestable containers for collections of document information. In SGML, containers are called *elements*. The DTD rule for the occurrence and sequence of document data and other elements inside a particular *kind* of element, or *element type*, is called a *content model*. Our recipe DTD creates six element types, and declares their content models as follows:

- A “recipe” element must contain a “title” element, followed by an “ingredient list” element, followed by an “instruction list” element. All these inner elements are required.
- A “title” element contains characters.
- An “ingredient list” element must contain one or more “ingredient” elements. An “ingredient” element contains characters.
- An “instruction list” element must contain one or more “step” elements. A “step” element contains characters.

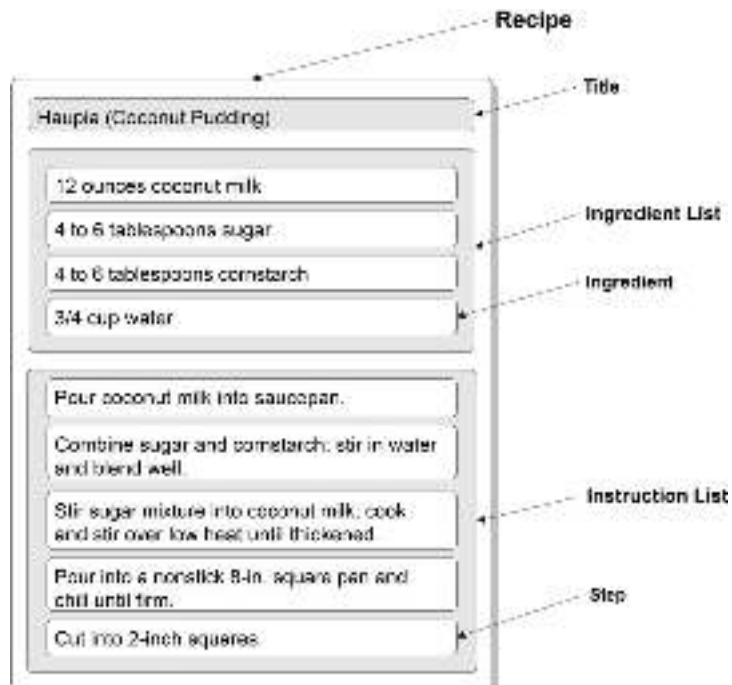


Figure 8.2. Recipe Elements

If you scan the recipe from top to bottom, you cross the upper and lower boundaries of each rectangle in the same logical places that you would come across SGML element markup in a document instance. Each upper boundary corresponds to an element *start-tag*, and each lower boundary to an element *end-tag*. By default, SGML tag markup consists of the name of the element type surrounded by angle brackets (<>), with the addition of a slash (/) before the name in the end-tag, as follows.

<recipe>	<i>recipe start-tag</i>
<title>	<i>title start-tag</i>
Haupia (Coconut Pudding)	
</title>	<i>title end-tag</i>
<ingredient-list>	<i>ingredient-List start-tag</i>
<ingredient>	<i>ingredient start-tag</i>
12 ounces coconut milk	
</ingredient>	<i>ingredient end-tag</i>
<ingredient>	
4 to 6 tablespoons sugar	
</ingredient>	
<ingredient>	
4 to 6 tablespoons cornstarch	
</ingredient>	

```

¾ cup water
</ingredient>
</ingredient-list>                                ingredient-list end-tag
<instruction-list>                                instruction-list start-tag
<step>                                             step start-tag
Pour coconut milk into saucepan.
</step>                                           step end-tag
<step>
Combine sugar and cornstarch; stir in water and blend well.
</step>
<step>
Stir sugar mixture into coconut milk;
cook and stir over low heat until thickened.
</step>
<step>
Pour into a nonstick 8-in. square pan and chill until firm.
</step>
<step>
Cut into 2-inch squares.
</step>
</instruction-list>                               instruction-list end-tag
</recipe>                                         recipe end-tag

```

Attributes

A DTD can specify rules for special labels, called *attributes*, that can be attached to particular elements to further describe their content. Our sample DTD declares that the recipe and step element types have attributes as follows.

- A “recipe” element can optionally have values for the following attributes:
 - Type of dish: The value can be any character string (for example, “starter” or “main course”).
 - Number of servings it makes: The value must be a number (for example, “10”).
 - Number of minutes it takes to prepare: The value must be a number (for example, “30”).

- A “step” element can optionally have a value indicating whether performing the step is necessary. The value can be either “yes” or “no”; if a value isn't supplied explicitly, the default value is “yes.”

```
<recipe type="dessert" servings="6" preptime="10"> recipe start-tag with attributes
.....
  <step necessary="no"> step start-tag with explicitly set value
the pot you will use
  </step>
.....
</recipe>
```

Entities

A DTD can identify fragments of document content, called *entities*, that are stored separately from the main content of the documents they're used in. Storing such a fragment separately allows it to be used multiple times and to be updated easily through the changing of a single definition. Documents use *entity references* to include an entity's content everywhere it is supposed to appear.

Our sample DTD creates an entity called “pour-chill-cut” that contains all the data and the entire markup for two sequential steps that are likely to occur in several recipes:

- A step with a default attribute value, containing the string “Pour into a nonstick 8-in. square pan and chill until firm.”
- A step with a default attribute value, containing the string “Cut into 2-inch squares.”

```
<step>
Stir sugar mixture into coconut milk;
cook and stir over low heat until thickened.
</step>
&pour-chill-cut;           reference to entity
</instruction-list>
</recipe>
```

Comments

Any SGML document can contain *comments*, notes to the author or to other readers of the SGML data and markup, in nearly any location. The DTD can't control whether or where comments are used, and the logical SGML “view” of the document ignores comments entirely as if they weren't present they appear in the SGML document instance, but they are disposed of during processing.

SGML Document Processing

A DTD is an essential part of an SGML-based document-processing environment. However, it is only a small part. To convert, create, and format documents, build databases

for search and retrieval, and otherwise manipulate your SGML documents effectively, you must use software applications. You can break down the kinds of software processing you need to perform into three major categories:

- **Document creation:** This category might include, for example, writing and revising data and markup using an SGML-aware text editor or word processor converting documents in non-SGML file formats to make them conform to a target DTD on a one-time or routine basis and assembling whole documents from fragments.
- **Document management:** This category might include, for example, storing and archiving documents and document fragments in a database extracting document fragments for assembly and controlling and tracking workflow, document revisions, and access to files.
- **Document utilization:** This category might include, for example, formatting documents for printing and online viewing; indexing them for online retrieval adding hyperlinks to them for online navigation and interchanging them with business partners and customers in original SGML form.

9. INTERNET PROTOCOL AND APPLICATIONS

- A protocol is a set of rules. Internet protocols are sets of rules governing communication within and between computers on a network. Protocol specifications define the format of the messages that are exchanged. A letter sent through the postal system also uses protocols. Part of the protocol specifies where on the envelope the delivery address needs to be written. If the delivery address is written in the wrong place, the letter cannot be delivered.
- Timing is crucial to network operation. Protocols require messages to arrive within a certain amount of time so that computers do not wait indefinitely for messages that may have been lost. Therefore, systems maintain one or more timers during transmission of data. Protocols also initiate alternative actions if the network does not meet the timing rules. Many protocols consist of a suite of other protocols that are stacked in layers. These layers depend on the operation of the other layers in the suite to function properly.
- These are the main functions of protocols:
 - Identifying errors
 - Compressing the data
 - Deciding how the data is to be sent
 - Addressing the data
 - Deciding how to announce sent and received data
 - Although many other protocols exist, Table 8-1 summarizes the functions of some of the more common protocols used on networks and the Internet.

Protocols	Function
TCP/IP	Transports data on the Internet
NetBEUI/NetBIOS	A small, fast protocol designed for a workgroup network that
	requires no connection to the Internet
IPX/SPX	Transports data on a Novell NetWare network
http/https	Defines how files are exchanged on the web
FTP	Provides services for file transfer and manipulation

SSH	Connects computers securely
Telnet	Uses a text-based connection to a remote TCP/IP computer
POP3	Downloads e-mail messages from an e-mail server
IMAP	Downloads e-mail messages from an e-mail server
SMTP	Sends mail in a TCP/IP network

To understand how networks and the Internet work, you must be familiar with the commonly used protocols. These protocols are used to browse the web, send and receive e-mail, and transfer data files. You will encounter other protocols as your experience in IT grows, but they are not used as often as the common protocols described here:

- **TCP/IP:** The TCP/IP suite of protocols has become the dominant standard for internetworking. TCP/IP represents a set of public standards that specify how packets of information are exchanged between computers over one or more networks.
- **IPX/SPX:** Internetwork Packet Exchange/Sequenced Packet Exchange is the protocol suite originally employed by Novell Corporation's network operating system, NetWare. It delivers functions similar to those included in TCP/IP. Novell in its current releases supports the TCP/IP suite. A large installed base of NetWare networks continues to use IPX/SPX.

- NetBEUI: NetBIOS Extended User Interface is a protocol used primarily on small Windows NT networks. NetBEUI cannot be routed or used by routers to talk to each other on a large network. NetBEUI is suitable for small peer-to-peer networks, involving a few computers directly connected to each other. It can be used in conjunction with another routable protocol such as TCP/IP. This gives the network administrator the advantages of the high performance of NetBEUI within the local network and the ability to communicate beyond the LAN over TCP/IP.
- AppleTalk: AppleTalk is a protocol suite to network Macintosh computers. It is composed of a comprehensive set of protocols that span the seven layers of the Open Systems Interconnection (OSI) reference model. The AppleTalk protocol was designed to run over LocalTalk, which is the Apple LAN physical topology. This protocol is also designed to run over major LAN types, notably Ethernet and Token Ring.
- HTTP: Hypertext Transfer Protocol governs how files such as text, graphics, sound, and video are exchanged on the World Wide Web (WWW). The Internet Engineering Task Force (IETF) developed the standards for HTTP.
- FTP: File Transfer Protocol provides services for file transfer and manipulation. FTP allows multiple simultaneous connections to remote file systems.
- SSH: Secure Shell is used to securely connect to a remote computer.
- Telnet: An application used to connect to a remote computer that lacks security features.
- POP3: Post Office Protocol is used to download e-mail from a remote mail server.
- IMAP: Internet Message Access Protocol is also used to download e-mail from a remote mail server.
- SMTP: Simple Mail Transfer Protocol is used to send e-mail to a remote e-mail server.
- The more you understand about each of these protocols, the more you will understand how networks and the Internet work.

UNIT I

INTRODUCTION TO INTERNET

1. INTERNET

The Internet is a worldwide system of interconnected computer networks. The computers and computer networks exchange information using TCP/IP (Transmission Control Protocol/Internet Protocol). The computers are connected via the telecommunications networks, and the Internet can be used for e-mailing, transferring files and accessing information on the World Wide Web.

- In 1969 the precursor of Internet is born: ARPAnet.
- ARPA = Advanced Research Projects Agency sponsored by the American Department of Defense (DOD). Designed to connect military research centers.
- Distributed computer system able to survive a nuclear attack.
- Problem: ARPAnet could connect only networks of the same type.
- In 1970, ARPA starts developing the Transmission Control Protocol / Internet Protocol (TCP/IP), a technology for connecting networks of different types (produced by different companies).
- Other networks appear, such as CSNET and BITNET.

2. SERVICE PROVIDED BY THE INTERNET

a) Electronic Mail (E-mail)

- Electronic mail, commonly known as email or e-mail, 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.
- Email servers accept, forward, deliver and store messages.
- An email message consists of three components, the message envelope, the message header, and the message body. The message header contains control information, including attachment option, reason box, email address and one or more recipient addresses. Message can consist of attachments, graphic or video/audio clips.
- E.g.: - e-mail addresses • Samsung@gmail.com • Apple@outlook.com
- Some popular E-mail services providers are: • 1. Gmail • 2. Hotmail • 3. Yahoo • 4. MSN

b) World wide web

World Wide Web (WWW) Important features of the world wide web (www) are listed below,

- Most important service provided by Internet.
- An internet-based hypermedia initiative for global information sharing.
- Developed in 1989 by Tim Berners-Lee of the European Particle Physics Lab (CERN) in Switzerland.

c) File Transfer Protocol

- File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet.
- FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.

d) Chat Rooms

- Real time typed conversation via computers.
- Chat rooms (the channel or medium).
- Chat clients (program used to connect to a chat server)
- Normally included on a browser
- Freely downloaded from the web
- Some are text only, others support voice & video

e) Mailing list

A mailing list is a collection of names and addresses used by an individual or an organization to send material to multiple recipients. The term is often extended to include the people subscribed to such a list, so the group of subscribers is referred to as "the mailing list", or simply "the list".

- Group of e-mail address given a single name
- When a message is sent to the mailing list everyone on the list receive the message
- To add your name to a mailing list you must subscribe to it, to remove your name you must unsubscribe.

f) Instant Messaging

Instant Messaging Instant messaging (IM) is a type of online chat which offers real-time text transmission over the Internet. A LAN messenger operates in a similar way over a local area network.

- Notifies you when one or more people are online allow exchange of messages and files
- It allows you to join a private chat rooms.
- Real time conversation that takes place on a computer
- Chat room is location on server that permits users to discuss topics of interest
- Some are the text only others support voice and video

g) News groups

A news group is called as forum, an on-line discussion group. On the Internet, there are literally thousands of newsgroups covering every conceivable interest. To view and post messages to a newsgroup, you need a news reader, a program that runs on your computer and connects you to a news server on the Internet.

- Online area in which users conduct written discussion about a particular subject.
- Usenet (collection of all internet news groups).
- News server (computer storing newsgroups messages).
- Newsreader (program used to access newsgroups).

3. INTERNET PRINCIPLES

Basic Internet Principles

- TCP/IP
- UDP
- IP Addresses
- Domain Names
- The Domain Name System (DNS)
- Ports
- Sockets
- URLs

TCP/IP

The Internet is the network that connects computers all over the world. It works according to a set of agreed-upon 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. The IP address of the sender (or source, or origin)
3. The IP address of recipient (or 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. Ensures delivery of data in the correct order
- IP is concerned with *routing*. IP attaches the address of the destination of each packet. IP ensures that packets get to the right place.
 - TCP is the higher-level protocol that uses the lower-level IP.
 - When an application is written, the general principle is to use the highest level protocol that you can, provided that it provides the functionality and performance that is required.

Many applications can be written using TCP/IP. For example, a Web browser can be written in Java using only URLs, without any explicit mention of sockets.

- On each machine an application program makes calls on procedures in the transport layer (normally TCP). In turn the transport layer makes calls on the Internet layer (normally IP). In turn the Internet layer makes calls on the physical layer, which is different depending on the technology of the communication link.
- At the destination machine, information is passed up through the layers to the application program. Each application program acts as if it is communicating directly with the application on another machine. The lower levels of the communication software and hardware are invisible.
- This four-layer model is sufficient for understanding Internet software. But there are other models that use a different number of layers, like the ISO seven-layer model.
- The application layer produces some data, adds a header to it and passes the complete package to the transport layer. The transport layer adds another header and passes the package to the internet layer. The internet layer adds another header and passes it to the physical layer. The application data is enclosed by 4 headers used by the different layers. This process can be thought of as repeatedly putting a letter into an envelope and then addressing the envelope.

UDP

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is **unreliable and connectionless protocol**. So, there is no need to establish connection prior to data transfer.

Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of Internet services, provides assured delivery, reliability and much more but all these services cost us with additional overhead and latency. Here, UDP comes into picture. For the real-time services like computer gaming, voice or video communication, live conferences, we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also saves bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

UDP Header –

UDP header is **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. First 8 Bytes contains all necessary header information and remaining part consists of data. UDP port number fields are each 16 bits long, therefore range for port numbers defined from 0 to 65535, port number 0 is reserved. Port numbers help to distinguish different user requests or process.

1. **Source Port** : Source Port is 2 Byte long field used to identify port number of source.
2. **Destination Port** : It is 2 Byte long field, used to identify the port of destined packet.
3. **Length** : Length is the length of UDP including header and the data. It is 16-bits field.
4. **Checksum** : Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, pseudo header of information from the IP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

IP Addresses

An *IP address* is a unique address for every host computer in the world. It consists of 4 bytes or 32 bits. This is represented in *quad notation* (or *dot notation*) as four 8-bit numbers, each in the range 0 to 255, e.g. 131.123.2.220.

IP addresses are registered so that they stay unique.

You can find the IP address of the local machine under Windows NT by typing the following command at the DOS prompt in a console window:

ipconfig

Under Unix or Linux, this command is:

ifconfig

The IP address 127.0.0.1 is a special address, called the *local loopback address* that denotes the local machine. A message sent to this address will simply return to the sender, without leaving the sender. It is useful for testing purposes.

Domain Names

A *domain name* is the user-friendly equivalent of an IP address. It is used because the numbers in an IP address are hard to remember and use. It is also known as a *host name*.

Example:

cs.stmarys.ca

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 is something like com, org, edu, ca, 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.

The Domain Name System

A program, say a Web browser, that wants to use 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 located in one place, so it is held in a *distributed database*.

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, or 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 generally 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 of peer-to-peer relationships: chat, internet phone, shared whiteboard.

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. A port number does not correspond to any physical connection on the machine, of which there might be just one. Each type of service has, by convention, a standard port number. Thus 80 usually mean 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.

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 at 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.

Streams

Accessing information across the Internet is accomplished using streams. A stream is a serial collection of data. An output stream can be sent to a printer, a display, a serial file, or an Internet connection, for example. Likewise, an input stream can come from a keyboard, a serial file, or from an Internet connection. Thus reading or writing to another program across a network or the Internet is just like reading or writing to a serial file.

URL

A URL (*Uniform Resource Locator*):

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

A URL has this structure:

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

- Things in square brackets indicate that the item can be omitted.
- 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.

The first part of a URL is the particular protocol. Some commonly-used protocols are:

http	The service is the Web. The file is accessed using the HTTP protocol.
ftp	The service is file transfer protocol. The URL locates a file, a directory or an FTP server.
telnet	The service is remote login to a host. No file name is needed.
mailto	The service is e-mail.
news	The URL specifies a usenet newsgroup.
file	This locates a file on the local system. The server part of the URL is omitted.

A pathname (optional) specifies a directory (folder). The pathname is not the complete directory name, but is relative to some directory (folder) designated by the administrator as the

directory in which publicly-accessible files are held. It would be unusual for a server to make available its entire file system to clients.

The file name can either be a data file name or can specify an executable file that produces a valid HTML document as its output. A file name is often omitted. In this case, the server decides which file to use. Many servers send a default file from the directory specified in the path name - for example a file called default.html, index.html or welcome.html.

The **section** part of a URL (optional) specifies a named anchor in an HTML document. Such a place in a document is specified by an HTML entry like:

```
<a name="url"></a>
```

4. BASIC WEB CONCEPTS

Clients, servers, and sites

- On the Web, information is stored at **Web sites**.
- Access to the information at a Web site is managed by a **Web server** for the site.
- Users access the information using a **Web client**, which is also called a **browser**.

How information is stored: HTML

Information on the Web is stored in documents, using a language named **HTML** (HyperText Markup Language). Web clients interpret HTML and 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).

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 link in the document shown in their Web client. Documents and locations within documents are identified by an address that is called a Uniform Resource Locator, or **URL**.

This is an example of a URL:

```
http://www.sybase.com/products
```

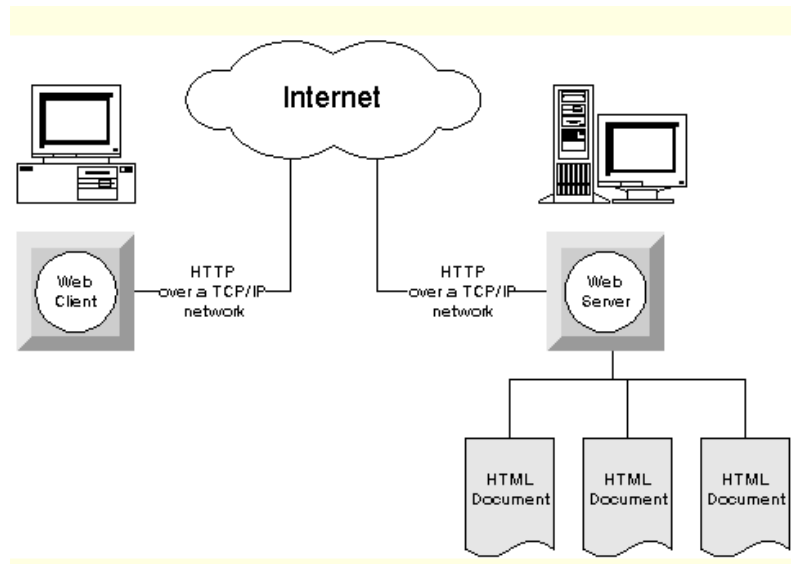


Figure 4.1 *Storing, locating and transmitting information on the web*

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 in 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 across the network to the Web client.

External data in HTML documents

HTML documents can reference external files (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.

Problems with file-based Web sites

In a file-based Web site, each document is a separate file. For large Web sites, this leads to management problems. For example, maintaining current copies of hundreds or thousands of different resources in separate files is difficult enough; maintaining the links between these resources is even more challenging.

Another problem is that many Web sites contain dynamic information - pricing information, for example, or employee information on an organization's intranet. Maintaining such information in HTML files in addition to the database where it resides is a huge task.

For these and other reasons, linking databases to the Web is increasingly the solution of choice for management of large Web sites and management of dynamic content. Database storage of Web information can either replace or complement file storage of Web resources.

5. CLIENT SERVER MODEL

Although an Internet system provides a basic communication service, the protocol software cannot initiate control with, or accept contact from, a remote computer.

Of course, two application involved in a communication **cannot both** wait for a message to arrive. One application must actively initiate interaction while the other application passively waits.

Most network applications use a form of communication known as **the client –server paradigm**. A server application waits passively for contact, while a client application initiates communication actively.

Client Server Model

A client and server networking model is a model in which computers such as servers provide the network services to the other computers such as clients to perform a user based tasks. This model is known as client-server networking model.

The application programs using the client-server model should follow the given below strategies:

- An application program is known as a client program, running on the local machine that requests for a service from an application program known as a server program, running on the remote machine.
- A client program runs only when it requests for a service from the server while the server program runs all time as it does not know when its service is required.
- A server provides a service for many clients not just for a single client. Therefore, we can say that client-server follows the many-to-one relationship. Many clients can use the service of one server.
- Services are required frequently, and many users have a specific client-server application program. For example, the client-server application program allows the user to access the files, send e-mail, and so on. If the services are more customized, then we should have one generic application program that allows the user to access the services available on the remote computer.

Process classification

- **Client process**, the process that requires a service
- **Server process**, the process that provides the required service

Client

A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the service started by the user and terminates when the service is completed.

Server

A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.

A server program is an infinite program means that when it starts, it runs infinitely unless the problem arises. The server waits for the incoming requests from the clients. When the request arrives at the server, then it responds to the request.

A server must guarantee:

- Authentication: client identity verification
- Authorization: verification of the possibility for a client to access to a particular service
- Data security: guarantee that specific data cannot be read and/or modified

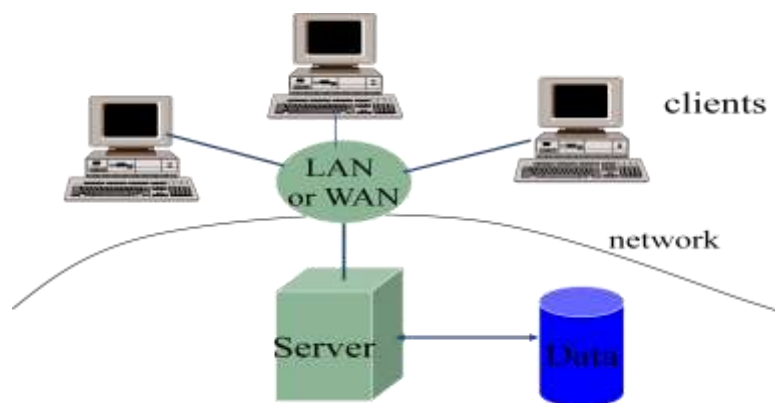


Figure 5.1 Architecture of Client Server Model

Establishing a Connection

- While establishing connection, both client and server will construct a socket.
- A socket is one end of an inter-process communication channel.
- The process to establish a socket on the client side is different than the process to establish a socket on the server side.

Establish a Socket on Client Side

- Create a socket using the `socket()` system call.
- Connect the socket to the address of the server using the `connect()` system call

- Send and receive data. You can use read() and write() system calls. However, there are different ways to send and receive data.

Establish a Socket on Client Side

- Create a socket using the socket() system call.
- Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number of the host machine.
- Listen for the connections using the listen() system call.
- Accept a connection using the accept() system call
- Send and receive data. You can use read() and write() system calls. However, there are different ways to send and receive data.

Address domain and the socket type

- The client and server processes can communicate with each other only if their sockets are in the same domain and of the same type.
- Two widely used address domains are:
 - UNIX domain (in which two processes share a common file system).
 - Internet domain (in which two processes run on any two hosts on the Internet)

UNIX and Internet domain

- The address of a socket in the Unix domain is a character string, an entry in the file system.
- The address of a socket in the Internet domain is the Internet address (IP address) of the host machine.
- Each socket needs a port number (16 bits unsigned integers) on the host.
- The lower numbers are reserved in Unix for standard services (e.g. 21 is used as the port number for the FTP server).
- Generally, port number above 2000 are available.

Types of Socket

- Two widely used socket types are:
 - Stream socket: treats communications as a continuous stream of characters. It uses TCP
 - Datagram socket: reads entire message at one time. It uses UDP.

Characteristics of Client-server architecture

- Client and server machines need different amount of hardware and software resources.
- Client and server machines may belong to different vendors.
- *Horizontal scalability* (increase of the client machines) and *vertical scalability* (migration to a more powerful server or to a multiserver solution)
- A client or server application interacts directly with a transport layer protocol to establish communication and to send or receive information.
- The transport protocol then uses lower layer protocols to send or receive individual messages. Thus, a computer needs a complete stack of protocols to run either a client or a server.

- A single server-class computer can offer multiple services at the same time; a separate server program is needed for each service.
- **Identifying a particular service** TCP uses 16-bit integer values (protocol port numbers) to identify services, and assign a unique port number to each service.
- A client specifies the protocol port number of the desired service when sending a request.

Advantages of Client-server networks:

- **Centralized:** Centralized back-up is possible in client-server networks, i.e., all the data is stored in a server.
- **Security:** These networks are more secure as all the shared resources are centrally administered.
- **Performance:** The use of the dedicated server increases the speed of sharing resources. This increases the performance of the overall system.
- **Scalability:** We can increase the number of clients and servers separately, i.e., the new element can be added, or we can add a new node in a network at any time.

Disadvantages of Client-Server network:

- **Traffic Congestion** is a big problem in Client/Server networks. When a large number of clients send requests to the same server may cause the problem of Traffic congestion.
- It does not have a robustness of a network, i.e., when the server is down, then the client requests cannot be met.
- A client/server network is very decisive. Sometimes, regular computer hardware does not serve a certain number of clients. In such situations, specific hardware is required at the server side to complete the work.
- Sometimes the resources exist in the server but may not exist in the client. For example, if the application is web, then we cannot take the print out directly on printers without taking out the print view window on the web.

6. HTML

HTML stands for Hypertext Markup Language, and it is the most widely used language to write Web Pages.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
- As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

HTML Tags

As told earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within angle braces **<Tag Name>**. Except few tags, most of the tags have their corresponding closing tags. For example, **<html>** has its closing tag **</html>** and **<body>** tag has its closing tag **</body>** tag etc.

Above example of HTML document uses the following tags –

Sr.No	Tag & Description
1	<!DOCTYPE...> This tag defines the document type and HTML version.
2	<html> This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
3	<head> This tag represents the document's header which can keep other HTML tags like <title> , <link> etc.
4	<title> The <title> tag is used inside the <head> tag to mention the document title.
5	<body> This tag represents the document's body which keeps other HTML tags like <h1> , <div> , <p> etc.
6	<h1> This tag represents the heading.
7	<p> This tag represents a paragraph.

HTML Document Structure

A typical HTML document will have the following structure –

```
<HTML>

    <head>

        HTML header related tags

    </head>

    <body>

        Document body related tags

    </body>

</HTML>
```

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration –

```
<!DOCTYPE html>
```

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Paragraph Tag

The **<p>** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening **<p>** and a closing **</p>** tag as shown below in the example –

Line Break Tag

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **
** tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use **
** it is not valid in XHTML.

Centering Content

You can use **<center>** tag to put any content in the center of the page or any table cell.

Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The `<hr>` tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag `<pre>`.

Any text between the opening `<pre>` tag and the closing `</pre>` tag will preserve the formatting of the source document.

HTML Elements

An **HTML element** is defined by a starting tag. If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags –

Start Tag	Content	End Tag
<code><p></code>	This is paragraph content.	<code></p></code>
<code><h1></code>	This is heading content.	<code></h1></code>
<code><div></code>	This is division content.	<code></div></code>
<code>
</code>		

So here `<p>...</p>` is an HTML element, `<h1>...</h1>` is another HTML element. There are some HTML elements which don't need to be closed, such as `<img.../>`, `<hr />` and `
` elements. These are known as **void elements**.

HTML Attributes

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts – a **name** and a **value**

- The **name** is the property you want to set. For example, the paragraph `<p>` element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.

- The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: **left**, **center** and **right**.

Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are –

- Id
- Title
- Class
- Style

The Id Attribute

The **id** attribute of an HTML tag can be used to uniquely identify any element within an HTML page. There are two primary reasons that you might want to use an id attribute on an element –

- If an element carries an id attribute as a unique identifier, it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

The title Attribute

The **title** attribute gives a suggested title for the element.

The class Attribute

The **class** attribute is used to associate an element with a style sheet, and specifies the class of element. You will learn more about the use of the class attribute when you will learn Cascading Style Sheet (CSS). So for now you can avoid it.

The style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

Generic Attributes

Here's a table of some other attributes that are readily usable with many of the HTML tags.

Attribute	Options	Function
align	right, left, center	Horizontally aligns tags

valign	top, middle, bottom	Vertically aligns tags within an HTML element.
bgcolor	numeric, hexadecimal, RGB values	Places a background color behind an element
background	URL	Places a background image behind an element
id	User Defined	Names an element for use with Cascading Style Sheets.
class	User Defined	Classifies an element for use with Cascading Style Sheets.
width	Numeric Value	Specifies the width of tables, images, or table cells.
height	Numeric Value	Specifies the height of tables, images, or table cells.
title	User Defined	"Pop-up" title of the elements.

HTML - Formatting

Anything that appears within **...** element, is displayed in bold.

- **Italic Text**

Anything that appears within *<i>...</i>* element is displayed in italicized

- **Underlined Text**

Anything that appears within <u>...</u> element, is displayed with underline

- **Superscript Text**

The content of a ^{^{...}} element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

- **Subscript Text**

The content of a `_{...}` element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

- **Larger Text**

The content of the `<big>...</big>` element is displayed one font size larger than the rest of the text surrounding it

- **Smaller Text**

The content of the `<small>...</small>` element is displayed one font size smaller than the rest of the text surrounding it

- **Grouping Content**

The `<div>` and `` elements allow you to group together several elements to create sections or subsections of a page.

For example, you might want to put all of the footnotes on a page within a `<div>` element to indicate that all of the elements within that `<div>` element relate to the footnotes. You might then attach a style to this `<div>` element so that they appear using a special set of style rules.

HTML – Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the `<table>` tag in which the `<tr>` tag is used to create table rows and `<td>` tag is used to create data cells. The elements under `<td>` are regular and left aligned by default

- **Table Heading**

Table heading can be defined using `<th>` tag. This tag will be put to replace `<td>` tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element in any row. Headings, which are defined in `<th>` tag are centered and bold by default.

- **Cell padding and Cell spacing Attributes**

There are two attributes called *cell padding* and *cell spacing* which you will use to adjust the white space in your table cells. The cell spacing attribute defines space between table cells, while cell padding represents the distance between cell borders and the content within a cell.

- **Colspan and Rowspan Attributes**

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

HTML - Text Links

- **Linking Documents**

A link is specified using HTML tag `<a>`. This tag is called **anchor tag** and anything between the opening `<a>` tag and the closing `` tag becomes part of the link and a user can click that part to reach to the linked document. Following is the simple syntax to use `<a>` tag.

- **The target Attribute**

We have used **target** attribute in our previous example. This attribute is used to specify the location where linked document is opened. Following are the possible options –

Sr.No	Option & Description
1	<code>_blank</code> Opens the linked document in a new window or tab.
2	<code>_self</code> Opens the linked document in the same frame.
3	<code>_parent</code> Opens the linked document in the parent frame.
4	<code>_top</code> Opens the linked document in the full body of the window.
5	<code>targetframe</code> Opens the linked document in a named <i>targetframe</i> .

Insert Image

You can insert any image in your web page by using **** tag. Following is the simple syntax to use this tag.

```
<img src = "Image URL"...attributes-list/>
```

The **** tag is an empty tag, which means that, it can contain only list of attributes and it has no closing tag.

7. SCRIPTING LANGUAGES

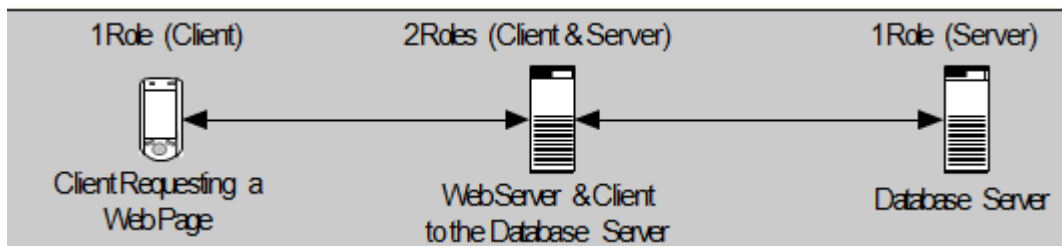
A scripting language is a programming language designed for integrating and communicating with other programming languages. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++.

There are many scripting languages some of them are discussed below:

- **Bash:** It is a scripting language to work in the Linux interface. It is a lot easier to use bash to create scripts than other programming languages. It describes the tools to use and code in the command line and creates useful reusable scripts and conserves documentation for other people to work with.
- **Node js:** It is a framework to write network applications using **JavaScript**. Corporate users of Node.js include IBM, LinkedIn, Microsoft, Netflix, PayPal, and Yahoo for real-time web applications.
- **Ruby:** There are a lot of reasons to learn Ruby programming language. Ruby's flexibility has allowed developers to create innovative software. It is a scripting language which is great for web development.
- **Python:** It is easy, free and open source. It supports procedure-oriented programming and object-oriented programming. Python is an interpreted language with dynamic semantics and a huge line of code are scripted and is currently the most hyped language among developers.
- **Perl:** A scripting language with innovative features to make it different and popular. Found on all windows and Linux servers. It helps in text manipulation tasks. High traffic websites that use Perl extensively include priceline.com, IMDB.
- **PHP:** PHP is a recursive acronym for "PHP: Hypertext Preprocessor". PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- **Javascript:** JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

client-side and server-side

- Any machine can play the role of either a client or a server
 - You could even have a machine being both
- Some languages, e.g. Javascript, are said to be *client-side*.
 - Run on the user's browser/web client
- Other languages, e.g. PHP, are said to be *server-side*.
 - Run on the server that is delivering content to the user



Servers

- Two important examples are:
 - Web Servers
 - Database Servers
 - Some of these machines may be powerful computers dedicated to the task, i.e. Database server
- A web server is a machine holding and delivering HTML pages that can be accessed by remote (client) machines over the Internet.
- **Static Web Model :** Client sends a request to the server for a web page. The server looks up the web page using part of the URL you have sent it, and then returns the HTML page which your browser subsequently displays on your machine.

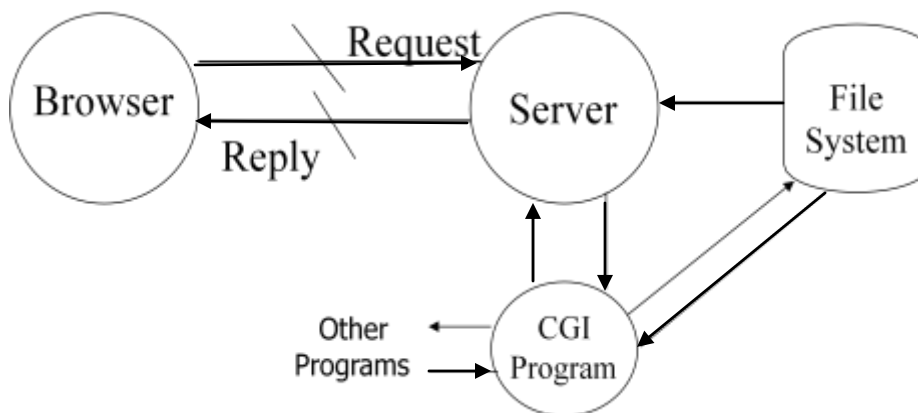


- **Dynamic model:** Client sends a request to the server and it dynamically determines the HTML that is to be returned.
- The dynamics of the reply is achieved through extending the *web server* with a program (script) that does some data processing and creates HTML output based on the data you

sent (e.g. contents of a form). The process of generating the HTML response is performed *server-side*.

SERVER SIDE SCRIPTING

- Dynamic web model
 - One approach is the Common Gateway Interface (CGI) where we have a separate program that can be executed.
 - An alternative is to have extra code in the HTML that can be executed on the *server* to determine the HTML that is to be returned.
 - That is how PHP works.



CLIENT SIDE SCRIPTING

- The other (complementary) approach is to do the work on the client machine.
 - Again we have extra code in the HTML, but now it is executed by the user's browser (i.e. *client-side*).
 - Most common client side script is Java script.
 - An example of its use is when a web page has a form. We can use Java script to validate the input data client-side before it is sent to a server.
 - If we do the validation on the client, this reduces the work that the server has to do and reduces the time taken to respond to the user.
- HTML5 essentially includes Java script elements to enhance its power.
- Java script can also be used to create dynamic web page content. For example:
 - We could change the content based on the fact that you visited the web page before.
 - Time of day.
 - JavaScript popup menus.

Simple Java script program

```
<html>

<head>

<title>A First Program in JavaScript</title>

</head>

<body>

<h1>Dynamic generation</h1>

<script language = "JavaScript">

    document.writeln("<p>Welcome to JavaScript programming</p>");

</script>

</body>

</html>
```

Advantages of scripting languages

- **Easy learning:** The user can learn to code in scripting languages quickly, not much knowledge of web technology is required.
- **Fast editing:** It is highly efficient with the limited number of data structures and variables to use.
- **Interactivity:** It helps in adding visualization interfaces and combinations in web pages. Modern web pages demand the use of scripting languages. To create enhanced web pages, fascinated visual description which includes background and foreground colors and so on.
- **Functionality:** There are different libraries which are part of different scripting languages. They help in creating new applications in web browsers and are different from normal programming languages.

Application of Scripting Languages

Scripting languages are used in many areas:

- Scripting languages are used in web applications. It is used in server side as well as client side. Server side scripting languages are: JavaScript, PHP, Perl etc. and client side scripting languages are: JavaScript, AJAX, jQuery etc.
- Scripting languages are used in system administration. For example: Shell, Perl, Python scripts etc.
- It is used in Games application and Multimedia.
- It is used to create plug-in and extensions for existing applications.

8. Standard Generalized Mark –up languages (SGML)

SGML is a language for recording and storing document information—a computer language that nonetheless can also be read and understood by humans. You use SGML to write rules that a group of related documents should follow when they store your desired added value. The general process of figuring out the rules is called *modeling*, and when you're done modeling and expressing the model in SGML form, your set of rules serves as the “language” that these documents use to “speak” to computers. In SGML terminology, such a group of documents is described by the term *document type*, and a rule set for a document type is called a *document type definition*, or *DTD*.

For example, newspapers might be a document type, and your SGML rules for adding information to the newspapers' electronic files would be a newspaper DTD. Such information might include, for instance, the date each feature article was assigned to be written and the name of the news organization from which each photo was obtained.

SGML, Document Types, and Documents

DTDs form the foundation for every SGML-based document production system by:

- Rigorously recording and enforcing your requirements for document intelligence and structure
- Controlling the text editors that insert and keep track of the added intelligence and structure, allowing some authoring functions to be automated
- Controlling the systems that manage whole and partial documents
- Providing information about the documents to the software that formats them, indexes them for retrieval, and otherwise processes them

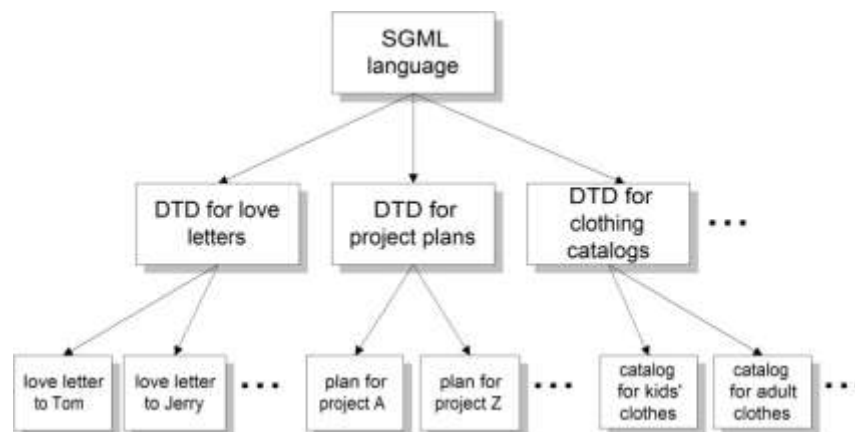


Figure 8.1. SGML, DTDs, and Document Instances

The SGML language can be used to express any number of DTDs. Likewise, each DTD can be used as the basis for many documents of the same general type, with each document being an *instance*—an example—of the type described by its DTD. Any set of similar documents can be considered a document type to be modeled using SGML DTD rules: love letters, project plans, product catalogs, and so on.

SGML Markup Strengths

To summarize, SGML markup is unique in that it combines several design strengths:

- It is declarative, which helps document producers “write once, use many”—putting the same document data to multiple uses, such as delivery of documents in a variety of online and paper formats and interchange with others who wish to use the documents in different ways.
- It is generic across systems and has a nonproprietary design, which helps make documents vendor and platform independent and “future-proof”—protecting them against changes in computer hardware and software.
- It is contextual, which heightens the quality and completeness of processing by allowing documents to be structurally validated and by enabling logical collections of data to be manipulated intelligently.

SGML Constructs

The grammar of every SGML markup language has four basic “parts of speech”: elements, attributes, entities, and comments. In the following sections we'll use a hypothetical set of DTD rules for a very simple “recipe” document type, along with two real recipes, to illustrate these parts of speech.

Elements

SGML and Other Markup Systems described the notion of nestable containers for collections of document information. In SGML, containers are called *elements*. The DTD rule for the occurrence and sequence of document data and other elements inside a particular *kind* of element, or *element type*, is called a *content model*. Our recipe DTD creates six element types, and declares their content models as follows:

- A “recipe” element must contain a “title” element, followed by an “ingredient list” element, followed by an “instruction list” element. All these inner elements are required.
- A “title” element contains characters.
- An “ingredient list” element must contain one or more “ingredient” elements. An “ingredient” element contains characters.
- An “instruction list” element must contain one or more “step” elements. A “step” element contains characters.

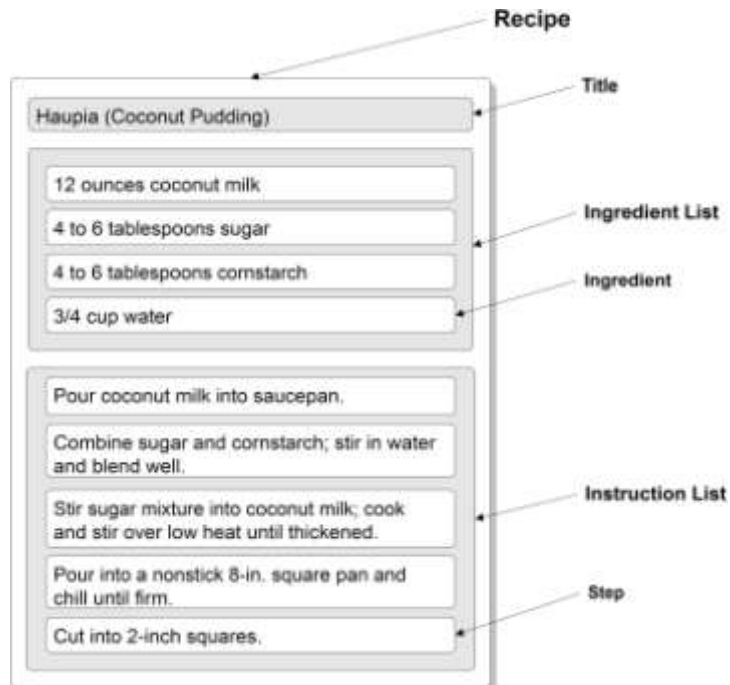


Figure 8.2. Recipe Elements

If you scan the recipe from top to bottom, you cross the upper and lower boundaries of each rectangle in the same logical places that you would come across SGML element markup in a document instance. Each upper boundary corresponds to an element *start-tag*, and each lower boundary to an element *end-tag*. By default, SGML tag markup consists of the name of the element type surrounded by angle brackets (<>), with the addition of a slash (/) before the name in the end-tag, as follows.

<recipe>	<i>recipe start-tag</i>
<title>	<i>title start-tag</i>
Haupia (Coconut Pudding)	
</title>	<i>title end-tag</i>
<ingredient-list>	<i>ingredient-List start-tag</i>
<ingredient>	<i>ingredient start-tag</i>
12 ounces coconut milk	
</ingredient>	<i>ingredient end-tag</i>
<ingredient>	
4 to 6 tablespoons sugar	
</ingredient>	
<ingredient>	
4 to 6 tablespoons cornstarch	
</ingredient>	

```

¾ cup water
</ingredient>
</ingredient-list>                                ingredient-list end-tag
<instruction-list>                                instruction-list start-tag
<step>                                             step start-tag
Pour coconut milk into saucepan.
</step>                                           step end-tag
<step>
Combine sugar and cornstarch; stir in water and blend well.
</step>
<step>
Stir sugar mixture into coconut milk;
cook and stir over low heat until thickened.
</step>
<step>
Pour into a nonstick 8-in. square pan and chill until firm.
</step>
<step>
Cut into 2-inch squares.
</step>
</instruction-list>                                instruction-list end-tag
</recipe>                                         recipe end-tag

```

Attributes

A DTD can specify rules for special labels, called *attributes*, that can be attached to particular elements to further describe their content. Our sample DTD declares that the recipe and step element types have attributes as follows.

- A “recipe” element can optionally have values for the following attributes:
 - Type of dish: The value can be any character string (for example, “starter” or “main course”).
 - Number of servings it makes: The value must be a number (for example, “10”).
 - Number of minutes it takes to prepare: The value must be a number (for example, “30”).

- A “step” element can optionally have a value indicating whether performing the step is necessary. The value can be either “yes” or “no”; if a value isn't supplied explicitly, the default value is “yes.”

```
<recipe type="dessert" servings="6" preptime="10"> recipe start-tag with attributes
.....
  <step necessary="no"> step start-tag with explicitly set value
the pot you will use
  </step>
.....
</recipe>
```

Entities

A DTD can identify fragments of document content, called *entities*, that are stored separately from the main content of the documents they're used in. Storing such a fragment separately allows it to be used multiple times and to be updated easily through the changing of a single definition. Documents use *entity references* to include an entity's content everywhere it is supposed to appear.

Our sample DTD creates an entity called “pour-chill-cut” that contains all the data and the entire markup for two sequential steps that are likely to occur in several recipes:

- A step with a default attribute value, containing the string “Pour into a nonstick 8-in. square pan and chill until firm.”
- A step with a default attribute value, containing the string “Cut into 2-inch squares.”

```
<step>
Stir sugar mixture into coconut milk;
cook and stir over low heat until thickened.
</step>
&pour-chill-cut;           reference to entity
</instruction-list>
</recipe>
```

Comments

Any SGML document can contain *comments*, notes to the author or to other readers of the SGML data and markup, in nearly any location. The DTD can't control whether or where comments are used, and the logical SGML “view” of the document ignores comments entirely as if they weren't present they appear in the SGML document instance, but they are disposed of during processing.

SGML Document Processing

A DTD is an essential part of an SGML-based document-processing environment. However, it is only a small part. To convert, create, and format documents, build databases

for search and retrieval, and otherwise manipulate your SGML documents effectively, you must use software applications. You can break down the kinds of software processing you need to perform into three major categories:

- **Document creation:** This category might include, for example, writing and revising data and markup using an SGML-aware text editor or word processor converting documents in non-SGML file formats to make them conform to a target DTD on a one-time or routine basis and assembling whole documents from fragments.
- **Document management:** This category might include, for example, storing and archiving documents and document fragments in a database extracting document fragments for assembly and controlling and tracking workflow, document revisions, and access to files.
- **Document utilization:** This category might include, for example, formatting documents for printing and online viewing; indexing them for online retrieval adding hyperlinks to them for online navigation and interchanging them with business partners and customers in original SGML form.

9. INTERNET PROTOCOL AND APPLICATIONS

- A protocol is a set of rules. Internet protocols are sets of rules governing communication within and between computers on a network. Protocol specifications define the format of the messages that are exchanged. A letter sent through the postal system also uses protocols. Part of the protocol specifies where on the envelope the delivery address needs to be written. If the delivery address is written in the wrong place, the letter cannot be delivered.
- Timing is crucial to network operation. Protocols require messages to arrive within a certain amount of time so that computers do not wait indefinitely for messages that may have been lost. Therefore, systems maintain one or more timers during transmission of data. Protocols also initiate alternative actions if the network does not meet the timing rules. Many protocols consist of a suite of other protocols that are stacked in layers. These layers depend on the operation of the other layers in the suite to function properly.
- These are the main functions of protocols:
 - Identifying errors
 - Compressing the data
 - Deciding how the data is to be sent
 - Addressing the data
 - Deciding how to announce sent and received data
 - Although many other protocols exist, Table 8-1 summarizes the functions of some of the more common protocols used on networks and the Internet.

Protocols	Function
TCP/IP	Transports data on the Internet
NetBEUI/NetBIOS	A small, fast protocol designed for a workgroup network that
	requires no connection to the Internet
IPX/SPX	Transports data on a Novell NetWare network
http/https	Defines how files are exchanged on the web
FTP	Provides services for file transfer and manipulation

SSH	Connects computers securely
Telnet	Uses a text-based connection to a remote TCP/IP computer
POP3	Downloads e-mail messages from an e-mail server
IMAP	Downloads e-mail messages from an e-mail server
SMTP	Sends mail in a TCP/IP network

To understand how networks and the Internet work, you must be familiar with the commonly used protocols. These protocols are used to browse the web, send and receive e-mail, and transfer data files. You will encounter other protocols as your experience in IT grows, but they are not used as often as the common protocols described here:

- **TCP/IP:** The TCP/IP suite of protocols has become the dominant standard for internetworking. TCP/IP represents a set of public standards that specify how packets of information are exchanged between computers over one or more networks.
- **IPX/SPX:** Internetwork Packet Exchange/Sequenced Packet Exchange is the protocol suite originally employed by Novell Corporation's network operating system, NetWare. It delivers functions similar to those included in TCP/IP. Novell in its current releases supports the TCP/IP suite. A large installed base of NetWare networks continues to use IPX/SPX.

- NetBEUI: NetBIOS Extended User Interface is a protocol used primarily on small Windows NT networks. NetBEUI cannot be routed or used by routers to talk to each other on a large network. NetBEUI is suitable for small peer-to-peer networks, involving a few computers directly connected to each other. It can be used in conjunction with another routable protocol such as TCP/IP. This gives the network administrator the advantages of the high performance of NetBEUI within the local network and the ability to communicate beyond the LAN over TCP/IP.
- AppleTalk: AppleTalk is a protocol suite to network Macintosh computers. It is composed of a comprehensive set of protocols that span the seven layers of the Open Systems Interconnection (OSI) reference model. The AppleTalk protocol was designed to run over LocalTalk, which is the Apple LAN physical topology. This protocol is also designed to run over major LAN types, notably Ethernet and Token Ring.
- HTTP: Hypertext Transfer Protocol governs how files such as text, graphics, sound, and video are exchanged on the World Wide Web (WWW). The Internet Engineering Task Force (IETF) developed the standards for HTTP.
- FTP: File Transfer Protocol provides services for file transfer and manipulation. FTP allows multiple simultaneous connections to remote file systems.
- SSH: Secure Shell is used to securely connect to a remote computer.
- Telnet: An application used to connect to a remote computer that lacks security features.
- POP3: Post Office Protocol is used to download e-mail from a remote mail server.
- IMAP: Internet Message Access Protocol is also used to download e-mail from a remote mail server.
- SMTP: Simple Mail Transfer Protocol is used to send e-mail to a remote e-mail server.
- The more you understand about each of these protocols, the more you will understand how networks and the Internet work.

XML

1. Introduction to XML

XML was designed to describe data.

HTML was designed to display data.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](#).

What is XML?

- XML stands for EXtensible Markup Language
 - XML is a markup language much like HTML
 - XML was designed to describe data, not to display data
 - XML tags are not predefined. You must define your own tags
 - XML is designed to be self-descriptive
 - XML is a W3C Recommendation
-

The Difference Between XML and HTML

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- XML was designed to describe data, with focus on what data is
- HTML was designed to display data, with focus on how data looks

HTML is about displaying information, while XML is about carrying information.

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

The following example is a note to Tove, from Jani, stored as XML:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

With XML You Invent Your Own Tags

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

XML is Not a Replacement for HTML

XML is a complement to HTML.

It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to describe data, while HTML is used to format and display the data.

My best description of XML is this:

XML is a software- and hardware-independent tool for carrying information.

2. How Can XML be Used?

XML is used in many aspects of web development, often to simplify data storage and sharing.

XML Separates Data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

XML Simplifies Data Sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc.), and make it more available for blind people, or people with other disabilities.

Internet Languages Written in XML

Several Internet languages are written in XML. Here are some examples:

- XHTML
- XML Schema
- SVG
- WSDL

RSS

3. XML Tree

An Example XML Document

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0).

The next line describes the **root element** of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

You can assume, from this example, that the XML document contains a note to Tove from Jani.

Don't you agree that XML is pretty self-descriptive?

XML Documents Form a Tree Structure

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

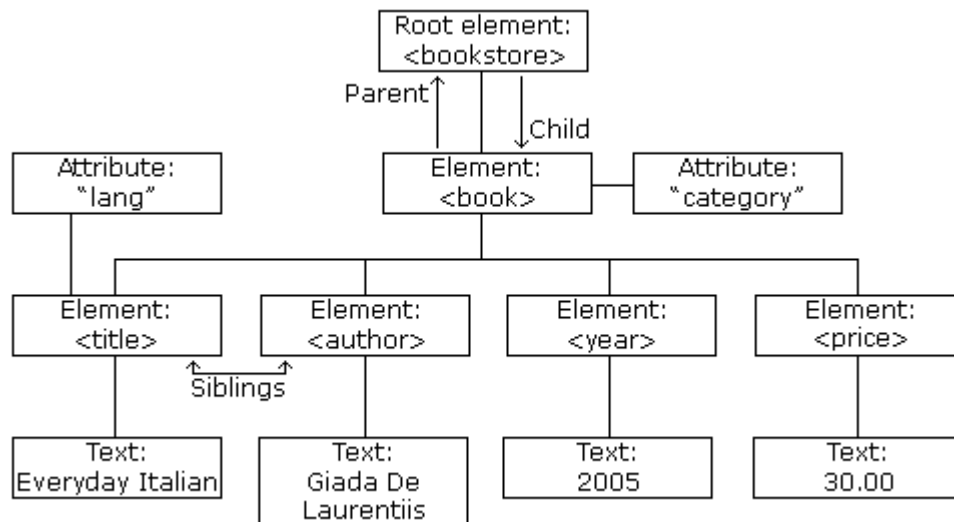
All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Example:



The image above represents one book in the XML below:

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is `<bookstore>`. All `<book>` elements in the document are contained within `<bookstore>`.

The <book> element has 4 children: <title>,< author>, <year>, <price>.

4. XML Syntax Rules

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

All XML Elements Must Have a Closing Tag

In HTML, some elements do not have to have a closing tag:

```
<p>This is a paragraph.  
<br>
```

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

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

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

```
<Message>This is incorrect</message>  
<message>This is correct</message>
```

Note: "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the example above, "Properly nested" simply means that since the `<i>` element is opened inside the `` element, it must be closed inside the `` element.

XML Documents Must Have a Root Element

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted.

INCORRECT:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

CORRECT:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Note: Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

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

White-space is Preserved in XML

XML does not truncate multiple white-spaces in a document (while HTML truncates multiple white-spaces to one single white-space):

XML: Hello Tove

HTML: Hello Tove

XML Stores New Line as LF

Windows applications store a new line as: carriage return and line feed (CR+LF).

Unix and Mac OSX uses LF.

Old Mac systems uses CR.

XML stores a new line as LF.

Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

5. XML Elements

An XML document contains XML Elements.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all of the above...

```
<bookstore>
<book category="CHILDREN">
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

```
<book category="WEB">
  <title>Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

Empty XML Elements

An element with no content is said to be empty.

In XML, you can indicate an empty element like this:

```
<element></element>
```

or you can use an empty tag, like this (this sort of element syntax is called self-closing):

```
<element />
```

The two forms above produce identical results in an XML parser.

Note: Empty elements do not have any content, but they can have attributes!

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Best Naming Practices

Create descriptive names, like this: <person>, <firstname>, <lastname>.

Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.

Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".

Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".

Avoid ":". Colons are reserved for namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

Naming Styles

There are no naming styles defined for XML elements. But here are some commonly used:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word except the first

If you choose a naming style, it is good to be consistent!

XML documents often have a corresponding database. A good practice is to use the naming rules of your database for the elements in the XML documents.

XML Elements are Extensible

XML elements can be extended to carry more information.

Look at the following XML example:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the `<to>`, `<from>`, and `<body>` elements from the XML document to produce this output:

MESSAGE

To: Tove

From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the `<to>`, `<from>`, and `<body>` elements in the XML document and produce the same output.

One of the beauties of XML, is that it can be extended without breaking applications.

6. XML Attributes

XML elements can have attributes, just like HTML.

Attributes provide additional information about an element.

XML Attributes

In HTML, attributes provide additional information about elements:

```
  
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

XML Attributes Must be Quoted

Attribute values must always be quoted. Either single or double quotes can be used. For a person's gender, the person element can be written like this:

```
<person gender="female">
```

or like this:

```
<person gender='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

XML Elements vs. Attributes

Take a look at these examples:

```
<person gender="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

```
<person>  
  <gender>female</gender>
```

```
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

In the first example gender is an attribute. In the last, gender is an element. Both examples provide the same information.

There are no rules about when to use attributes or when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead.

My Favorite Way

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="2008-01-10">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
  <date>2008-01-10</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third: (THIS IS MY FAVORITE):

```
<note>
  <date>
    <year>2008</year>
    <month>01</month>
    <day>10</day>
  </date>
  <to>Tove</to>
  <from>Jani</from>
```

```
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Avoid XML Attributes?

Some of the problems with using attributes are:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain. Use elements for data. Use attributes for information that is not relevant to the data.

Don't end up like this:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

XML Attributes for Metadata

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML. This example demonstrates this:

```
<messages>
<note id="501">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
<note id="502">
  <to>Jani</to>
  <from>Tove</from>
  <heading>Re: Reminder</heading>
  <body>I will not</body>
</note>
</messages>
```

The id attributes above are for identifying the different notes. It is not a part of the note itself.

What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

7. XML Namespaces

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a `<table>` element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a so-called **namespace** for the prefix must be defined.

The namespace is defined by the **xmlns attribute** in the start tag of an element.

The namespace declaration has the following syntax. *xmlns:prefix="URI"*.

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can be declared in the elements where they are used or in the XML root element:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">
```

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Note: The namespace URI is not used by the parser to look up information.

The purpose is to give the namespace a unique name. However, often companies use the namespace as a pointer to a web page containing namespace information.

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN).

In our examples we will only use URLs.

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

xmlns="namespaceURI"

This XML carries HTML table information:

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a piece of furniture:

```
<table xmlns="http://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Namespaces in Real Use

XSLT is an XML language that can be used to transform XML documents into other formats, like HTML.

In the XSLT document below, you can see that most of the tags are HTML tags.

The tags that are not HTML tags have the prefix `xsl`, identified by the namespace `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr>
          <th style="text-align:left">Title</th>
          <th style="text-align:left">Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
```

```
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

If you want to learn more about XSLT, please read our [XSLT Tutorial](#).

8. XML Encoding

XML documents can contain international characters, like Norwegian æøå, or French êëé.

To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.

Character Encoding

Character encoding defines a unique binary code for each different character used in a document.

In computer terms, character encoding are also called character set, character map, code set, and code page.

The Unicode Consortium

The Unicode Consortium develops the Unicode Standard. Their goal is to replace the existing character sets with its standard Unicode Transformation Format (UTF).

The Unicode Standard has become a success and is implemented in HTML, XML, Java, JavaScript, E-mail, ASP, PHP, etc. The Unicode standard is also supported in many operating systems and all modern browsers.

The Unicode Consortium cooperates with the leading standards development organizations, like ISO, W3C, and ECMA.

The Unicode Character Sets

Unicode can be implemented by different character sets. The most commonly used encodings are UTF-8 and UTF-16.

UTF-8 uses 1 byte (8-bits) to represent basic Latin characters, and two, three, or four bytes for the rest.

UTF-16 uses 2 bytes (16 bits) for most characters, and four bytes for the rest.

UTF-8 = The Web Standard

UTF-8 is the standard character encoding on the web.

UTF-8 is the default character encoding for HTML5, CSS, JavaScript, PHP, SQL, and XML.

XML Encoding

The first line in an XML document is called the **prolog**:

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

The prolog is optional. Normally it contains the XML version number.

It can also contain information about the encoding used in the document. This prolog specifies UTF-8 encoding:

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

The XML standard states that all XML software must understand both UTF-8 and UTF-16.

UTF-8 is the default for documents without encoding information.

In addition, most XML software systems understand encodings like ISO-8859-1, Windows-1252, and ASCII.

XML Errors

Most often, XML documents are created on one computer, uploaded to a server on a second computer, and displayed by a browser on a third computer.

If the encoding is not correctly interpreted by all the three computers, the browser might display meaningless text, or you might get an error message.

For high quality XML documents, UTF-8 encoding is the best to use. UTF-8 covers international characters, and it is also the default, if no encoding is declared.

Conclusion

When you write an XML document:

- Use an XML editor that supports encoding
- Make sure you know what encoding the editor uses
- Describe the encoding in the encoding attribute
- UTF-8 is the safest encoding to use
- UTF-8 is the web standard

9. Displaying XML

Raw XML files can be viewed in all major browsers.

Don't expect XML files to be displayed as HTML pages.

Viewing XML Files

```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Look at the XML file above in your browser: [note.xml](#)

Notice that an XML document will be displayed with color-coded root and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure. To view the raw XML source (without the + and - signs), select "View Page Source" or "View Source" from the browser menu.

Note: In Safari, only the element text will be displayed. To view the raw XML, you must right click the page and select "View Source".

Viewing an Invalid XML File

If an erroneous XML file is opened, some browsers report the error, and some only display it incorrectly.

Try to open the following XML file in Chrome, IE, Firefox, Opera, and Safari: [note_error.xml](#).

Other XML Examples

Viewing some XML documents will help you get the XML feeling:

[An XML CD catalog](#)

This is a CD collection, stored as XML.

[An XML plant catalog](#)

This is a plant catalog from a plant shop, stored as XML.

[An XML breakfast menu](#)

This is a breakfast food menu from a restaurant, stored as XML.

Why Does XML Display Like This?

XML documents do not carry information about how to display the data.

Since XML tags are "invented" by the author of the XML document, browsers do not know if a tag like <table> describes an HTML table or a dining table.

Without any information about how to display the data, most browsers will just display the XML document as it is.

Displaying XML Files with CSS?

Below is an example of how to use CSS to format an XML document.

We can use an XML file like [cd_catalog.xml](#) and a style sheet like [cd_catalog.css](#)

RESULT: [The CD catalog formatted with the CSS file](#)

Below is a fraction of the XML file. The second line links the XML file to the CSS file:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  .
  .
  .
</CATALOG>
```



Formatting XML with CSS is not recommended. Use JavaScript or XSLT instead.

10. XML Document Types

An XML document with correct syntax is called "Well Formed".

A "Valid" XML document must also conform to a document type definition.

Well Formed XML Documents

An XML document with correct syntax is "Well Formed".

The syntax rules were described in the previous chapters:

- XML documents must have a root element

- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

An XML Validator

To help you check the syntax of your XML files, we have created an [XML validator](#) to syntax-check your XML.

Valid XML Documents

A "valid" XML document is not the same as a "well formed" XML document.

A "valid" XML document must be well formed. In addition it must conform to a document type definition.

Rules that defines the legal elements and attributes for XML documents are called Document Type Definitions (DTD) or XML Schemas.

There are two different document type definitions that can be used with XML:

- DTD - The original Document Type Definition
 - XML Schema - An XML-based alternative to DTD
-

When to Use a DTD/Schema?

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

Your application can use a standard DTD to verify that the data you receive from the outside world is valid.

You can also use a DTD to verify your own data.

When to NOT to Use a DTD/Schema?

XML does not require a DTD/Schema.

When you are experimenting with XML, or when you are working with small XML files, creating DTDs may be a waste of time.

If you develop applications, wait until the specification is stable before you add a document definition. Otherwise, your software might stop working because of validation errors.

11. XML Validator

Use our XML validator to syntax-check your XML.

XML Errors Will Stop You

Errors in XML documents will stop your XML applications.

The W3C XML specification states that a program should stop processing an XML document if it finds an error. The reason is that XML software should be small, fast, and compatible.

HTML browsers will display HTML documents with errors (like missing end tags).

With XML, errors are not allowed.

Syntax-Check Your XML

To help you syntax-check your XML, we have created an XML validator.

Paste your XML into the text area below, and syntax-check it by clicking the "Validate" button.

```
<?xml version="1.0" encoding="UTF-8" ?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this wee
</note>
```

Syntax-Check an XML File

You can syntax-check an XML file by typing the URL of the file into the input field below, and then click the "Validate" button:

Filename:



If you get "Access Denied" or "Network Error", it is because your browser does not allow file access across domains.

The file "note_error.xml" demonstrates your browsers error handling. If you want to see an error-free message, substitute the "note_error.xml" with "cd_catalog.xml".

A General XML Validator

To help you check your xml files, you can [syntax-check any XML file](#) here.

Parse Errors

You can read more about the parse errors in our [XML DOM tutorial](#).

12. XML Schema

An XML Schema describes the structure of an XML document, just like a DTD.

An XML document with correct syntax is called "Well Formed".

An XML document validated against an XML Schema is both "Well Formed" and "Valid".

XML Schema

XML Schema is an XML-based alternative to DTD:

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

The Schema above is interpreted like this:

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string

Everything is wrapped in "Well Formed" XML.

XML Schemas are More Powerful than DTD

- XML Schemas are written in XML
 - XML Schemas are extensible to additions
 - XML Schemas support data types
 - XML Schemas support namespaces
-

Why Use an XML Schema?

With XML Schema, your XML files can carry a description of its own format.

With XML Schema, independent groups of people can agree on a standard for interchanging data.

With XML Schema, you can verify data.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types:

- It is easier to describe document content
 - It is easier to define restrictions on data
 - It is easier to validate the correctness of data
 - It is easier to convert data between different data types
-

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT

If you want to study XML Schema, please read our [XML Schema Tutorial](#).

Chapter 14. Dynamic HTML: Event Model

DHTML's *event model* lets scripts respond to user actions.

Event `onclick`

The `onclick` event fires when the user clicks the mouse.

The following causes the enclosed script to be executed when the element with id *element_ID* is clicked.

```
<script type = "text/javascript"
    for = "element_ID" event = "onclick">
    ...
</script>
```

We get inline scripting by including `onclick = "script"` in the opening tag of an element – e.g.,

```
<p onclick = "alert( 'second' )" >p2</p>
```

Example:

```
<html>

<head>
<title>onclick Example</title>

<script type = "text/javascript" for = "p1"
        event = "onclick">

    alert( "first" );

</script>

</head>

<body>

<p id = "p1">p1</p>

<p onclick = "alert( 'second' )" >p2</p>

</body>
</html>
```

The rendering is:

p1

p2

When **p1** is clicked, an alert box appears with text **first**.

When **p2** is clicked, an alert box appears with text **second**.

The alert boxes may be raised and dismissed any number of times.

The event Object, and Tracking the Mouse with Event onmouseover

The event object contains information about the triggered event.

Some of the properties of event are:

`type` is the name of the event that fired.

`srcElement` is a reference to the object that fired the event.

This object has its usual properties – e.g.,

`event.srcElement.tagName`

`propertyName` is the name of the property that changed in this event.

`offsetX` / `offsetY` are the coordinates of the mouse cursor relative to the object that fired the event.

`x` / `y` are the coordinates of the mouse cursor relative to the parent element of the element that fired the event.

`screenX` / `screenY` are the coordinates of the mouse cursor on the screen coordinate system.

`clientX` / `clientY` are the coordinates of the mouse cursor within the client area (the active area where the page is displayed).

`altKey` is true if the *ALT* key was pressed when the event fired.

Similar properties: `ctrlKey` and `shiftKey`.

`button` gives the mouse button pressed: 1 (left), 2 (right), 3 (left and right), 4 (middle), 5 (left and middle), 6 (right and middle), or 7 (all three).

See Figure 14.5, p. 464 in the text for more.

Event `onmousemove` fires constantly when the mouse is in motion.

Example:

```
<html>

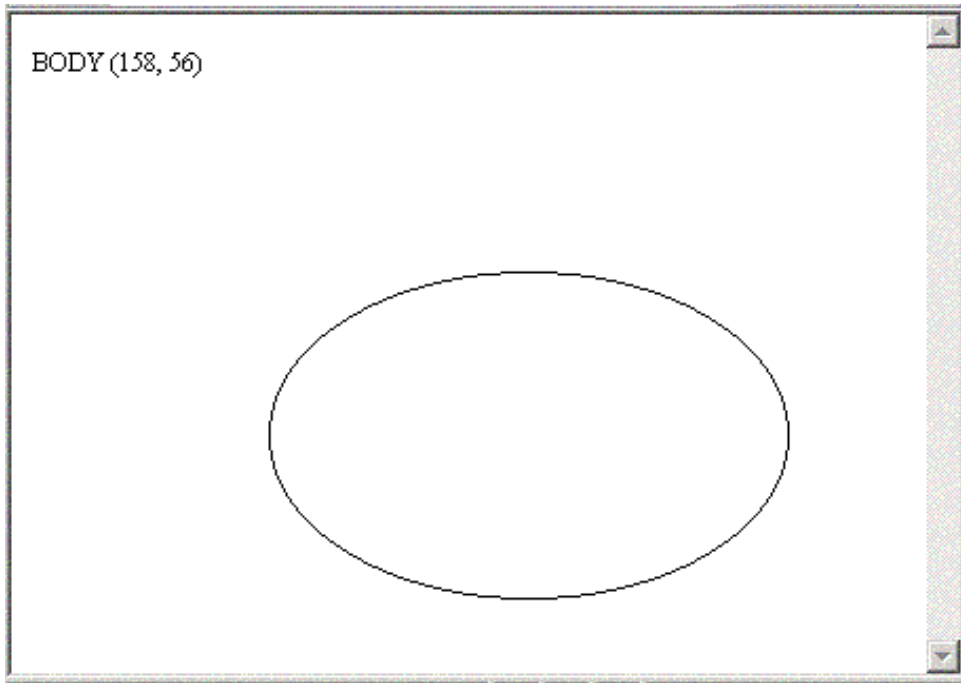
<head>
<title>onmousemove example</title>
<script type = "text/javascript">
    function updateMouseCoordinates()
    {
        coordinates.innerText =
            event.srcElement.tagName +
            " (" + event.offsetX + ", " +
            event.offsetY + ")";
    }
</script>
</head>

<body onmousemove = "updateMouseCoordinates()">

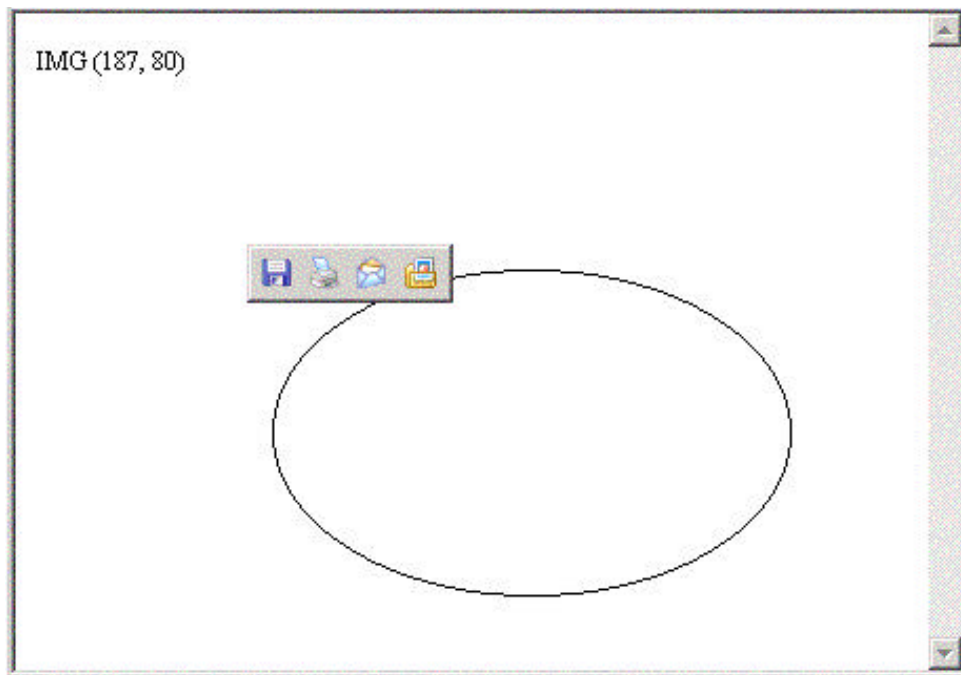
<span id = "coordinates">(0, 0)</span><br>
<img src = "ellipse.bmp"
    style ="position: absolute; top: 100;
        left: 100">

</body>
```

When the mouse cursor is not over the image, the coordinates are in the body's coordinate system.



When it's over the image, the coordinates are in the image's coordinate system.



Rollovers with `onmouseover` and `onmouseout`

When the mouse cursor moves over an element, event `onmouseover` is fired for that element.

When the mouse cursor leaves an element, event `onmouseout` is fired for that element.

Handling these events, we can achieve a *rollover* effect: the text of an element changes as the mouse moves over and out of it.

The document object has predefined event-handler slots for certain events.

You can associate an event handler with an event by assigning its name to the corresponding slot:

```
document.event_handler_slot = event_handler;
```

Then, when the corresponding event happens, *event_handler* is invoked.

The name *event_handler_slot* is usually the lowercase event name.

Examples:

```
document.onmouseover = mOverHandler;
document.onmouseout  = mOutHandler;
```

Here `mOverHandler` and `mOutHandler` are event handlers for the events `onmouseover` and `onmouseout` written by the programmer.

The following is needed for the next example.

We can create a new image object and set its `src` property to the file containing the image with

```
variable_name = new Image( );
variable_name.src = "file_name" ;
```

Creating an image object lets us pre-load the desired image rather than delaying download to when the image is displayed.

Example:

Here we define in the body a 2×2 table (where we show the `id`'s in parentheses):

1 (E0)	2 (E1)
3 (E2)	4 (E3)

In the script, we initialize two array variables:

```
ar1  = [ "one", "two", "three", "four" ],
ar2  = [ "I", "II", "III", "IV" ]
```

When the mouse cursor moves over the cell with `id Ei`, the numeral word `ar1[i]` appears in the cell.

When the mouse cursor moves out of the cell, the Roman numeral `ar2[i]` appears in it.

The name (a string) of the of the source element of the event is given by

```
event.srcElement.id
```

For this example, we use all but the first character of this as a subscript into either the `ar1` or the `ar2` array:

```
ar1[ event.srcElement.id.substr( 1 ) ]
```

Both event handlers first check whether the event was the image object (with `id "first"`).

If not, they check that the source element of the event has an `id` (that the `id` property is not undefined):

```
if ( event.srcElement.id )
```

We must do this since both events (`onmouseover` and `onmouseout`) fire for any element, but we process these events only for certain elements (those with `id`'s).

```
<html>
<head>
<title>Rollover Example</title>
<script type = "text/javascript">
    var  image1 = new Image(),
        image2 = new Image(),
        ar1     = [ "one", "two", "three", "four" ],
        ar2     = [ "I", "II", "III", "IV" ];

    image1.src = "ellipse.bmp";
    image2.src = "rectangle.bmp";

    document.onmouseover = mOverHandler;
    document.onmouseout  = mOutHandler;

    function mOverHandler()
    {
        if ( event.srcElement.id == "first" )
            event.srcElement.src = image2.src;
        else if (event.srcElement.id )
            event.srcElement.innerHTML =
                ar1[ event.srcElement.id.substr(1) ];
    }

    function mOutHandler()
    {
        if ( event.srcElement.id == "first" )
            event.srcElement.src = image1.src;
        else if (event.srcElement.id )
            event.srcElement.innerHTML =
                ar2[ event.srcElement.id.substr(1) ];
    }
</script>
</head>
```

Continued next page

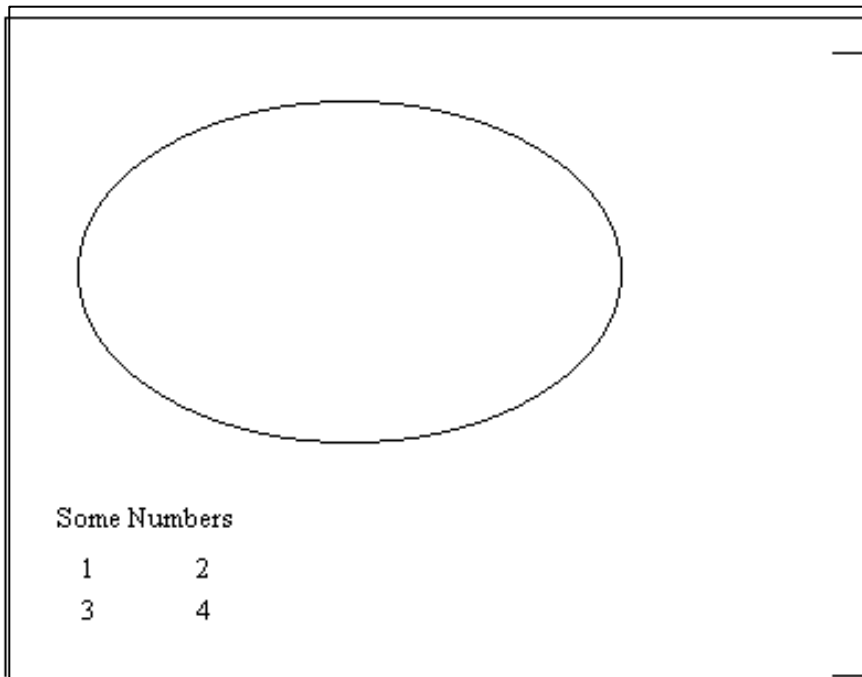
Continued from previous page

```
<body>

<img src = "ellipse.bmp" id = "first">

<table style = "width: 30%; text-align: center">
  <caption>Some Numbers</caption>
  <tr style = "text-align: center">
    <td id = E0 style= "width: 15%">1</td>
    <td id = E1 style= "width: 15%">2</td>
  </tr>
  <tr style = "text-align: center">
    <td id = E2 style= "width: 15%">3</td>
    <td id = E3 style= "width: 15%">4</td>
  </tr>
</table>

</body>
</html>
```



When the mouse cursor enters the area with the image of the ellipse, the image of a rectangle appears.

When the mouse cursor leaves the area with the image of the rectangle, the image of ellipse re-appears.

These images keep swapping: when the mouse cursor is in the area, we have a square; when it's out, we have an ellipse.

When the mouse cursor enters a cell with one of the numbers, say, **2**, it is replaced with its number word, **two**.

When the mouse cursor leaves this cell, the Roman number **II** appears.

Two and **II** keep swapping: when the mouse cursor is in the cell, we have **two**; when it's out, we have **II**.

But **2** never re-appears.

Error Handling with `onerror`

Scripts can use the `onerror` event to execute specialized error-handling code in place of the error dialog presented by browsers.

Suppose we define a function `errorHandler` as the handler for the `onerror` event:

```
document.onerror = errorHandler;
```

The header of this function should have three arguments (the names are up to you):

```
function errorHandler( errType, errURL,  
                      errLineNum )
```

Here

- `errType` (a string) is the type of the error,
- `errURL` (a string) is URL of the page where the error occurred, and
- `errLineNum` (an integer) is the number of the line where the error occurred.

It returns `true` to indicate that it has handled the error.

If it returns `false`, the default response (an error dialog) occurs.

The following is needed for the next example.

We can cause text to appear in the status bar at the bottom of the browser window by assigning it to `window.status`.

Example

We define an error handler that writes to the status bar the type of error and the line and URL where it occurred.

We force an error by associating a call to an undefined function with the `onclick` event of a paragraph element.

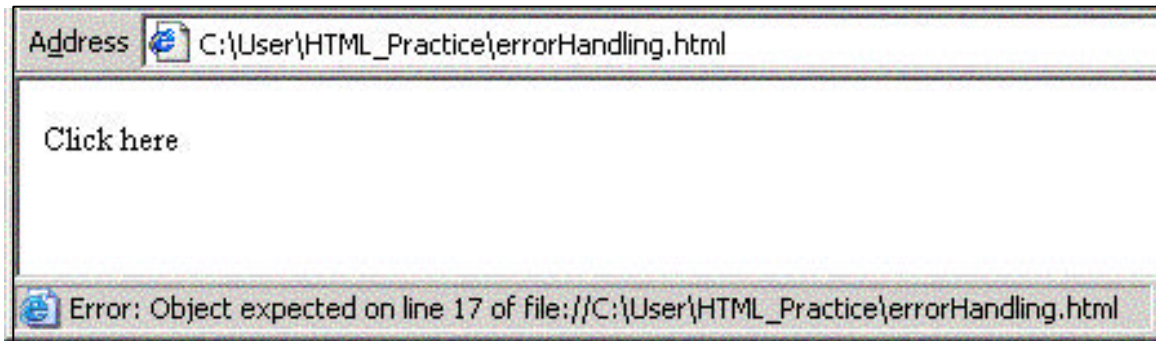
```
<html>
<head>
<title>onerror event</title>
<script type = "text/javascript">
    document.onerror = errorHandler;

    function errorHandler( errType, errURL,
                           errLineNum )
    {
        window.status = "Error: " + errType +
                        " on line " + errLineNum
                        + " of " + errURL;

        return true;
    }
</script>
</head>

<body>
    <p onclick = "doThis()">Click here</p>
</body>
</html>
```

When the text **Click here** of the paragraph is clicked, we get:



To make sure this example works, open the Control Panel and select **Internet Options**.

Click the **Advanced** tab.

Under **Browsing**, make sure the

Disable script debugging

check box is not selected.

Event Bubbling

Suppose an element and one of its ancestors both respond to a given event.

Then, without provision to the contrary, the event “bubbles” up from the element to the ancestor.

Suppose you want the event to be handled by the given element’s event handler and not by that of the ancestor.

Then you can set the event’s `cancelBubble` property:

```
event.cancelBubble = true;
```

Example:

Here the document has a handler for event `onclick`:

```
document.onclick = documentClick;
```

Both paragraphs in the body use `paragraphClick(b_value)`, defined in the script, as their handler for `onclick`.

One has `true` as the argument, which cancels bubbling.

The other has `false`, permitting bubbling.

```
<html>

<head>
<title>Event Bubbling</title>

<script type = "text/javascript">
    function documentClick()
    {
        alert( "You clicked in the document" );
    }

    function paragraphClick( value )
    {
        alert( "You clicked the text" );
        if ( value )
            event.cancelBubble = true;
    }

    document.onclick = documentClick;
</script>
</head>

<body>

<p onclick = "paragraphClick( false )">
    Click here!</p>

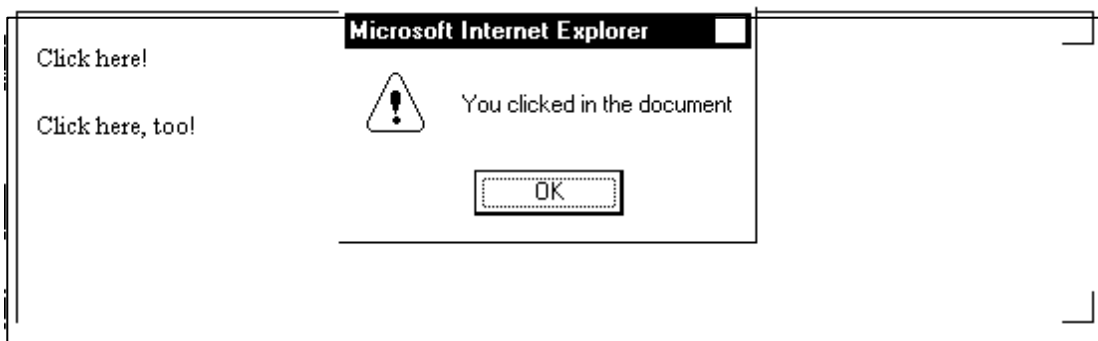
<p onclick = "paragraphClick( true )">
    Click here, too!</p>

</body>
</html>
```

The following shows the screen after the text **Click here!** was clicked, and after the alert box stating

You clicked the text
was dismissed.

The `onclick` event has bubbled up from the paragraph to the document.



If the text **Click here, too!** is clicked, then an alert box stating
You clicked the text
again appears.

But now, when this alert box is dismissed, no further alert box appears – bubbling up to the document has been canceled.

More DHTML Events

See Figure 14.10, pp. 474-476, in the text for descriptions of the remaining events, not discussed in detail in Chapter 14.



CSS - Quick Guide

Advertisements

Previous Page

Next Page

What is CSS?

- **Superior styles to HTML** – CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Multiple Device Compatibility** – Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- **Global web standards** – Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.

Who Creates and Maintains CSS?

CSS is created and maintained through a group of people within the W3C called the CSS Working Group. The CSS Working Group creates documents called specifications. When a specification has been discussed and officially ratified by the W3C members, it becomes a recommendation.

These ratified specifications are called recommendations because the W3C has no control over the actual implementation of the language. Independent companies and organizations create that software.

NOTE – The World Wide Web Consortium, or W3C is a group that makes recommendations about how the Internet works and how it should evolve.

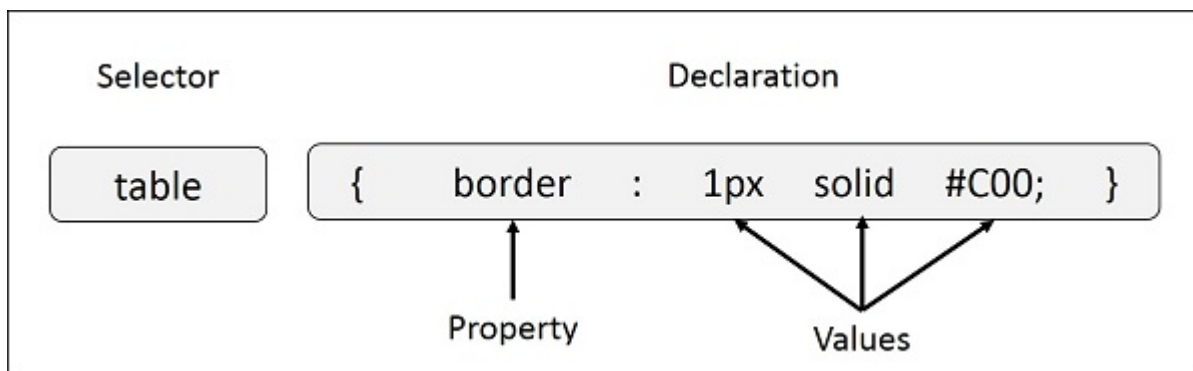
CSS Versions

Cascading Style Sheets level 1 (CSS1) came out of W3C as a recommendation in December 1996. This version describes the CSS language as well as a simple visual formatting model for all the HTML tags.



CSS - Syntax

selector { property: value }



Example – You can define a table border as follows –

```
table{ border :1px solid #C00; }
```

Here table is a selector and border is a property and given value *1px solid #C00* is the value of that property.

You can define selectors in various simple ways based on your comfort. Let me put these selectors one by one.

The Type Selectors

This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings –

```
h1 {  
  color: #36CFFF;  
}
```

The Universal Selectors

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type –

```
* {  
  color: #000000;  
}
```



Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, style rule will apply to `` element only when it lies inside `` tag.

```
ul em {  
    color: #000000;  
}
```

The Class Selectors

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

```
.black {  
    color: #000000;  
}
```

This rule renders the content in black for every element with class attribute set to *black* in our document. You can make it a bit more particular. For example –

```
h1.black {  
    color: #000000;  
}
```

This rule renders the content in black for only `<h1>` elements with class attribute set to *black*.

You can apply more than one class selectors to given element. Consider the following example –

```
<p class = "center bold">  
    This para will be styled by the classes center and bold.  
</p>
```

The ID Selectors

You can define style rules based on the *id* attribute of the elements. All the elements having that *id* will be formatted according to the defined rule.

```
#black {  
    color: #000000;  
}
```

This rule renders the content in black for every element with *id* attribute set to *black* in our document. You can make it a bit more particular. For example –



This rule renders the content in black for only `<h1>` elements with *id* attribute set to *black*.

The true power of *id* selectors is when they are used as the foundation for descendant selectors, For example –

```
#black h2 {  
    color: #000000;  
}
```

In this example all level 2 headings will be displayed in black color when those headings will lie with in tags having *id* attribute set to *black*.

The Child Selectors

You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example –

```
body > p {  
    color: #000000;  
}
```

This rule will render all the paragraphs in black if they are direct child of `<body>` element. Other paragraphs put inside other elements like `<div>` or `<td>` would not have any effect of this rule.

The Attribute Selectors

You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of *text* –

```
input[type = "text"] {  
    color: #000000;  
}
```

The advantage to this method is that the `<input type = "submit" />` element is unaffected, and the color applied only to the desired text fields.

There are following rules applied to attribute selector.

p[lang] – Selects all paragraph elements with a *lang* attribute.

p[lang="fr"] – Selects all paragraph elements whose *lang* attribute has a value of exactly "fr".

p[lang~="fr"] – Selects all paragraph elements whose *lang* attribute contains the word "fr".

p[lang|="en"] – Selects all paragraph elements whose *lang* attribute contains values that are exactly "en", or begin with "en-".



combine multiple properties and corresponding values into a single block as defined in the following example –

```
h1 {  
  color: #36C;  
  font-weight: normal;  
  letter-spacing: .4em;  
  margin-bottom: 1em;  
  text-transform: lowercase;  
}
```

Here all the property and value pairs are separated by a **semicolon (;)**. You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines.

For a while, don't bother about the properties mentioned in the above block. These properties will be explained in the coming chapters and you can find complete detail about properties in CSS References

Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example –

```
h1, h2, h3 {  
  color: #36C;  
  font-weight: normal;  
  letter-spacing: .4em;  
  margin-bottom: 1em;  
  text-transform: lowercase;  
}
```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

You can combine the various *id* selectors together as shown below –

```
#content, #footer, #supplement {  
  position: absolute;  
  left: 510px;  
  width: 200px;  
}
```

CSS - Inclusion



It will produce the following result –

This is a heading

This is a paragraph.

Attributes

Attributes associated with <style> elements are –

Attribute	Value	Description
type	text/css	Specifies the style sheet language as a content-type (MIME type). This is required attribute.
media	screen tty tv projection handheld print braille aural all	Specifies the device the document will be displayed on. Default value is <i>all</i> . This is an optional attribute.

Inline CSS - The *style* Attribute

You can use *style* attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax –

```
<element style = "...style rules....">
```

Attributes

Attribute	Value	Description



Example

Following is the example of inline CSS based on the above syntax –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <h1 style = "color:#36C;">
      This is inline CSS
    </h1>
  </body>
</html>
```

It will produce the following result –

This is inline CSS

External CSS - The <link> Element

The <link> element can be used to include an external stylesheet file in your HTML document.

An external style sheet is a separate text file with **.css** extension. You define all the Style rules within this text file and then you can include this file in any HTML document using <link> element.

Here is the generic syntax of including external CSS file –

```
<head>
  <link type = "text/css" href = "... " media = "... " />
</head>
```

Attributes

Attributes associated with <style> elements are –

Attribute	Value	Description
type	text css	Specifies the style sheet language as a content-type (MIME type). This attribute is required.



media	screen tty tv projection handheld print braille aural all	Specifies the device the document will be displayed on. Default value is <i>all</i> . This is optional attribute.
-------	---	---

Example

Consider a simple style sheet file with a name *mystyle.css* having the following rules –

```
h1, h2, h3 {
  color: #36C;
  font-weight: normal;
  letter-spacing: .4em;
  margin-bottom: 1em;
  text-transform: lowercase;
}
```

Now you can include this file *mystyle.css* in any HTML document as follows –

```
<head>
  <link type = "text/css" href = "mystyle.css" media = " all" />
</head>
```

Imported CSS - @import Rule

@import is used to import an external stylesheet in a manner similar to the <link> element. Here is the generic syntax of @import rule.

```
<head>
  <@import "URL";
</head>
```

Here URL is the URL of the style sheet file having style rules. You can use another syntax as well –

```
<head>
  <@import url("URL");
```



Example

Following is the example showing you how to import a style sheet file into HTML document –

```
<head>
  @import "mystyle.css";
</head>
```

CSS Rules Overriding

We have discussed four ways to include style sheet rules in a an HTML document. Here is the rule to override any Style Sheet Rule.

Any inline style sheet takes highest priority. So, it will override any rule defined in <style>...</style> tags or rules defined in any external style sheet file.

Any rule defined in <style>...</style> tags will override rules defined in any external style sheet file.

Any rule defined in external style sheet file takes lowest priority, and rules defined in this file will be applied only when above two rules are not applicable.

Handling old Browsers

There are still many old browsers who do not support CSS. So, we should take care while writing our Embedded CSS in an HTML document. The following snippet shows how you can use comment tags to hide CSS from older browsers –

```
<style type = "text/css">
  <!--
    body, td {
      color: blue;
    }
  -->
</style>
```

CSS Comments

Many times, you may need to put additional comments in your style sheet blocks. So, it is very easy to comment any part in style sheet. You can simple put your comments inside /*.....this is a comment in style sheet.....*/.

You can use /**/ to comment multi-line blocks in similar way you do in C and C++ programming languages.

Example



```

<head>
  <style>
    p {
      color: red;
      /* This is a single-line comment */
      text-align: center;
    }
    /* This is a multi-line comment */
  </style>
</head>

<body>
  <p>Hello World!</p>
</body>
</html>

```

It will produce the following result –

Hello World!

CSS - Measurement Units

% Defines a measurement as a percentage relative to another value, typically an enclosing element. p {font-size: 16pt; line-height: 125%;} cm Defines a measurement in centimeters. div {margin-bottom: 2cm;} em A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 12pt; thus, 2em would be 24pt. p {letter-spacing: 7em;} ex This value defines a measurement relative to a font's x-height. The x-height is determined by the height of the font's lowercase letter x. p {font-size: 24pt; line-height: 3ex;} in Defines a measurement in inches. p {word-spacing: .15in;} mm Defines a measurement in millimeters. p {word-spacing: 15mm;} pc Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch. p {font-size: 20pc;} pt Defines a measurement in points. A point is defined as 1/72nd of an inch. body {font-size: 18pt;} px Defines a measurement in screen pixels. p {padding: 25px;}

CSS - Colors

Hex Code #RRGGBB p{color:#FF0000;} Short Hex Code #RGB p{color:#6A7;} RGB % rgb(rrr%,ggg%,bbb%) p{color:rgb(50%,50%,50%);} RGB Absolute rgb(rrr,ggg,bbb) p{color:rgb(0,0,255);} keyword aqua, black, etc. p{color:teal;}

These formats are explained in more detail in the following sections –



next two are a green value(GG), and the last are the blue value(BB).

A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Jasc Paintshop Pro, or even using Advanced Paint Brush.

Each hexadecimal code will be preceded by a pound or hash sign '#'. Following are the examples to use Hexadecimal notation.

Color	Color HEX
	#000000
	#FF0000
	#00FF00
	#0000FF
	#FFFF00
	#00FFFF
	#FF00FF
	#C0C0C0
	#FFFFFF

CSS Colors - Short Hex Codes

This is a shorter form of the six-digit notation. In this format, each digit is replicated to arrive at an equivalent six-digit value. For example: #6A7 becomes #66AA77.

A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Jasc Paintshop Pro, or even using Advanced Paint Brush.

Each hexadecimal code will be preceded by a pound or hash sign '#'. Following are the examples to use Hexadecimal notation.

Color	Color HEX
	#000
	#F00
	#0F0
	#0FF



	#F0F
	#FFF

CSS Colors - RGB Values

This color value is specified using the **rgb()** property. This property takes three values, one each for red, green, and blue. The value can be an integer between 0 and 255 or a percentage.

NOTE – All the browsers does not support rgb() property of color so it is recommended not to use it.

Following is the example to show few colors using RGB values.

Color	Color RGB
	rgb(0,0,0)
	rgb(255,0,0)
	rgb(0,255,0)
	rgb(0,0,255)
	rgb(255,255,0)
	rgb(0,255,255)
	rgb(255,0,255)
	rgb(192,192,192)
	rgb(255,255,255)

Building Color Codes

You can build millions of color codes using our Color Code Builder. Check our **HTML Color Code Builder**. To use this tool, you would need a Java Enabled Browser.

Browser Safe Colors

Here is the list of 216 colors which are supposed to be most safe and computer independent colors. These colors vary from hexa code 000000 to FFFFFF. These colors are safe to use because they ensure that all computers would display the colors correctly when running a 256 color palette –

000000	000033	000066	000099	0000CC	0000FF



009900	009933	009966	009999	0099CC	0099FF
00CC00	00CC33	00CC66	00CC99	00CCCC	00CCFF
00FF00	00FF33	00FF66	00FF99	00FFCC	00FFFF
330000	330033	330066	330099	3300CC	3300FF
333300	333333	333366	333399	3333CC	3333FF
336600	336633	336666	336699	3366CC	3366FF
339900	339933	339966	339999	3399CC	3399FF
33CC00	33CC33	33CC66	33CC99	33CCCC	33CCFF
33FF00	33FF33	33FF66	33FF99	33FFCC	33FFFF
660000	660033	660066	660099	6600CC	6600FF
663300	663333	663366	663399	6633CC	6633FF
666600	666633	666666	666699	6666CC	6666FF
669900	669933	669966	669999	6699CC	6699FF
66CC00	66CC33	66CC66	66CC99	66CCCC	66CCFF
66FF00	66FF33	66FF66	66FF99	66FFCC	66FFFF
990000	990033	990066	990099	9900CC	9900FF
993300	993333	993366	993399	9933CC	9933FF
996600	996633	996666	996699	9966CC	9966FF
999900	999933	999966	999999	9999CC	9999FF
99CC00	99CC33	99CC66	99CC99	99CCCC	99CCFF
99FF00	99FF33	99FF66	99FF99	99FFCC	99FFFF
CC0000	CC0033	CC0066	CC0099	CC00CC	CC00FF
CC3300	CC3333	CC3366	CC3399	CC33CC	CC33FF
CC6600	CC6633	CC6666	CC6699	CC66CC	CC66FF



CCFF00	CCFF33	CCFF66	CCFF99	CCFFCC	CCFFFF
FF0000	FF0033	FF0066	FF0099	FF00CC	FF00FF
FF3300	FF3333	FF3366	FF3399	FF33CC	FF33FF
FF6600	FF6633	FF6666	FF6699	FF66CC	FF66FF
FF9900	FF9933	FF9966	FF9999	FF99CC	FF99FF
FFCC00	FFCC33	FFCC66	FFCC99	FFCCCC	FFCCFF
FFFF00	FFFF33	FFFF66	FFFF99	FFFFCC	FFFFFF

CSS - Backgrounds

Set the Background Color

Following is the example which demonstrates how to set the background color for an element.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "background-color:yellow;">
      This text has a yellow background color.
    </p>
  </body>
</html>
```

This will produce following result –

This text has a yellow background color.

Set the Background Image

We can set the background image by calling local stored images as shown below –



```
<style>
  body {
    background-image: url("/css/images/css.jpg");
    background-color: #cccccc;
  }
</style>
</head>

<body>
  <h1>Hello World!</h1>
</body>
<html>
```

It will produce the following result –

Hello World!

Repeat the Background Image

The following example demonstrates how to repeat the background image if an image is small. You can use *no-repeat* value for *background-repeat* property if you don't want to repeat an image, in this case image will display only once.

By default *background-repeat* property will have *repeat* value.

[Live Demo](#)

```
<html>
  <head>
    <style>
      body {
        background-image: url("/css/images/css.jpg");
        background-repeat: repeat;
      }
    </style>
  </head>

  <body>
    <p>Tutorials point</p>
  </body>
</html>
```

It will produce the following result –



The following example which demonstrates how to repeat the background image vertically.

[Live Demo](#)

```
<html>
  <head>
    <style>
      body {
        background-image: url("/css/images/css.jpg");
        background-repeat: repeat-y;
      }
    </style>
  </head>

  <body>
    <p>Tutorials point</p>
  </body>
</html>
```

It will produce the following result –



The following example demonstrates how to repeat the background image horizontally.

[Live Demo](#)

```
<html>
  <head>
    <style>
      body {
        background-image: url("/css/images/css.jpg");
        background-repeat: repeat-x;
      }
    </style>
  </head>

  <body>
    <p>Tutorials point</p>
  </body>
</html>
```

It will produce the following result –

Tutorials point



the left side.

[Live Demo](#)

```
<html>
  <head>
    <style>
      body {
        background-image: url("/css/images/css.jpg");
        background-position:100px;
      }
    </style>
  </head>

  <body>
    <p>Tutorials point</p>
  </body>
</html>
```

It will produce the following result –

Tutorials point

The following example demonstrates how to set the background image position 100 pixels away from the left side and 200 pixels down from the top.

[Live Demo](#)

```
<html>
  <head>
    <style>
      body {
        background-image: url("/css/images/css.jpg");
        background-position:100px 200px;
      }
    </style>
  </head>
```



It will produce the following result –

Tutorials point

Set the Background Attachment

Background attachment determines whether a background image is fixed or scrolls with the rest of the page.

The following example demonstrates how to set the fixed background image.

Live Demo

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-image: url('/css/images/css.jpg');
        background-repeat: no-repeat;
        background-attachment: fixed;
      }
    </style>
  </head>

  <body>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
```



It will produce the following result –

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The following example demonstrates how to set the scrolling background image.

Live Demo

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {
        background-image: url('/css/images/css.jpg');
        background-repeat: no-repeat;
        background-attachment: fixed;
        background-attachment: scroll;
      }
    </style>
  </head>

  <body>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
    <p>The background-image is fixed. Try to scroll down the page.</p>
  </body>
</html>
```



</html>

It will produce the following result –

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

The background-image is fixed. Try to scroll down the page.

Shorthand Property

You can use the *background* property to set all the background properties at once. For example –

```
<p style = "background:url(/images/pattern1.gif) repeat fixed;">
  This paragraph has fixed repeated background image.
</p>
```

CSS - Fonts

Set the Font Family

Following is the example, which demonstrates how to set the font family of an element. Possible value could be any font family name.

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-family:georgia,garamond,serif;">
      This text is rendered in either georgia, garamond, or the
      default serif font depending on which font you have at your system.
    </p>
```

[Live Demo](#)



This will produce following result –

This text is rendered in either georgia, garamond, or the default serif font depending on which font you have at your system.

Set the Font Style

Following is the example, which demonstrates how to set the font style of an element. Possible values are *normal*, *italic* and *oblique*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-style:italic;">
      This text will be rendered in italic style
    </p>
  </body>
</html>
```

This will produce following result –

This text will be rendered in italic style

Set the Font Variant

The following example demonstrates how to set the font variant of an element. Possible values are *normal* and *small-caps*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-variant:small-caps;">
      This text will be rendered as small caps
    </p>
  </body>
</html>
```



Set the Font Weight

The following example demonstrates how to set the font weight of an element. The font-weight property provides the functionality to specify how bold a font is. Possible values could be *normal*, *bold*, *bolder*, *lighter*, *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800*, *900*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-weight:bold;">
      This font is bold.
    </p>

    <p style = "font-weight:bolder;">
      This font is bolder.
    </p>

    <p style = "font-weight:500;">
      This font is 500 weight.
    </p>
  </body>
</html>
```

This will produce following result –

This font is bold.

This font is bolder.

This font is 500 weight.

Set the Font Size

The following example demonstrates how to set the font size of an element. The font-size property is used to control the size of fonts. Possible values could be *xx-small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*, *size in pixels* or *in %*.

[Live Demo](#)

```
<html>
  <head>
```



```
<p style = font-size:20px; >
  This font size is 20 pixels
</p>

<p style = "font-size:small;">
  This font size is small
</p>

<p style = "font-size:large;">
  This font size is large
</p>
</body>
</html>
```

This will produce following result –

This font size is 20 pixels

This font size is small

This font size is large

Set the Font Size Adjust

The following example demonstrates how to set the font size adjust of an element. This property enables you to adjust the x-height to make fonts more legible. Possible value could be any number.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-size-adjust:0.61;">
      This text is using a font-size-adjust value.
    </p>
  </body>
</html>
```

This will produce following result –

This text is using a font-size-adjust value.



the user's computer to have an expanded or condensed version of the font being used.

Possible values could be *normal*, *wider*, *narrower*, *ultra-condensed*, *extra-condensed*, *condensed*, *semi-condensed*, *semi-expanded*, *expanded*, *extra-expanded*, *ultra-expanded*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font-stretch:ultra-expanded;">
      If this doesn't appear to work, it is likely that your computer
      doesn't have a <br>condensed or expanded version of the font being used.
    </p>
  </body>
</html>
```

This will produce following result –

If this doesn't appear to work, it is likely that your computer doesn't have a condensed or expanded version of the font being used.

Shorthand Property

You can use the *font* property to set all the font properties at once. For example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "font:italic small-caps bold 15px georgia;">
      Applying all the properties on the text at once.
    </p>
  </body>
</html>
```

This will produce following result –

APPLYING ALL THE PROPERTIES ON THE TEXT AT ONCE.



- The **text-decoration** property is used to underline, overline, and strikethrough text.
- The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.
- The **white-space** property is used to control the flow and formatting of text.
- The **text-shadow** property is used to set the text shadow around a text.

Set the Text Color

The following example demonstrates how to set the text color. Possible value could be any color name in any valid format.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "color:red;">
      This text will be written in red.
    </p>
  </body>
</html>
```

It will produce the following result –

This text will be written in red.

Set the Text Direction

The following example demonstrates how to set the direction of a text. Possible values are *ltr* or *rtl*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
```



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

It will produce the following result –

This text will be rendered from right to left

Set the Space between Characters

The following example demonstrates how to set the space between characters. Possible values are *normal* or a number specifying space..

[Live Demo](#)

```
<html>  
  <head>  
  </head>  
  
  <body>  
    <p style = "letter-spacing:5px;">  
      This text is having space between letters.  
    </p>  
  </body>  
</html>
```

It will produce the following result –

T h i s t e x t i s h a v i n g s p a c e b e t w e e n l e t t e r s .

Set the Space between Words

The following example demonstrates how to set the space between words. Possible values are *normal* or a number specifying space.

[Live Demo](#)

```
<html>  
  <head>  
  </head>  
  
  <body>  
    <p style = "word-spacing:5px;">  
      This text is having space between words.  
    </p>  
  </body>  
</html>
```



This text is having space between words.

Set the Text Indent

The following example demonstrates how to indent the first line of a paragraph. Possible values are *%* or *a number specifying indent space*.

[Live Demo](#)

```
<html>
<head>
</head>

<body>
  <p style = "text-indent:1cm;">
    This text will have first line indented by 1cm and this line will remain at
    its actual position this is done by CSS text-indent property.
  </p>
</body>
</html>
```

It will produce the following result –

This text will have first line indented by 1cm and this line will remain at its actual position this is done by CSS text-indent property.

Set the Text Alignment

The following example demonstrates how to align a text. Possible values are *left*, *right*, *center*, *justify*.

[Live Demo](#)

```
<html>
<head>
</head>

<body>
  <p style = "text-align:right;">
    This will be right aligned.
  </p>

  <p style = "text-align:center;">
    This will be center aligned.
  </p>
```



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

This will produce following result –

This will be right aligned.

This will be center aligned.

This will be left aligned.

Decorating the Text

The following example demonstrates how to decorate a text. Possible values are *none*, *underline*, *overline*, *line-through*, *blink*.

[Live Demo](#)

```
<html>  
  <head>  
  </head>  
  
  <body>  
    <p style = "text-decoration:underline;">  
      This will be underlined  
    </p>  
  
    <p style = "text-decoration:line-through;">  
      This will be striked through.  
    </p>  
  
    <p style = "text-decoration:overline;">  
      This will have a over line.  
    </p>  
  
    <p style = "text-decoration:blink;">  
      This text will have blinking effect  
    </p>  
  </body>  
</html>
```

This will produce following result –



This will have a over line.

This text will have blinking effect

Set the Text Cases

The following example demonstrates how to set the cases for a text. Possible values are *none*, *capitalize*, *uppercase*, *lowercase*.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "text-transform:capitalize;">
      This will be capitalized
    </p>

    <p style = "text-transform:uppercase;">
      This will be in uppercase
    </p>

    <p style = "text-transform:lowercase;">
      This will be in lowercase
    </p>
  </body>
</html>
```

This will produce following result –

This Will Be Capitalized

THIS WILL BE IN UPPERCASE

this will be in lowercase

Set the White Space between Text

The following example demonstrates how white space inside an element is handled. Possible values are *normal*, *pre*, *nowrap*.

[Live Demo](#)



```
<body>
  <p style = "white-space:pre;">
    This text has a line break and the white-space pre setting
    tells the browser to honor it just like the HTML pre tag.
  </p>
</body>
</html>
```

This will produce following result –

This text has a line break and the white-space pre setting tells the browser to honor it just like the HTML pre tag.

Set the Text Shadow

The following example demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

[Live Demo](#)

```
<html>
<head>
</head>

<body>
  <p style = "text-shadow:4px 4px 8px blue;">
    If your browser supports the CSS text-shadow property,
    this text will have a blue shadow.
  </p>
</body>
</html>
```

It will produce the following result –

If your browser supports the CSS text-shadow property, this text will have a blue shadow.

CSS - Using Images



a value in length or in %.

A width of zero pixels means no border.

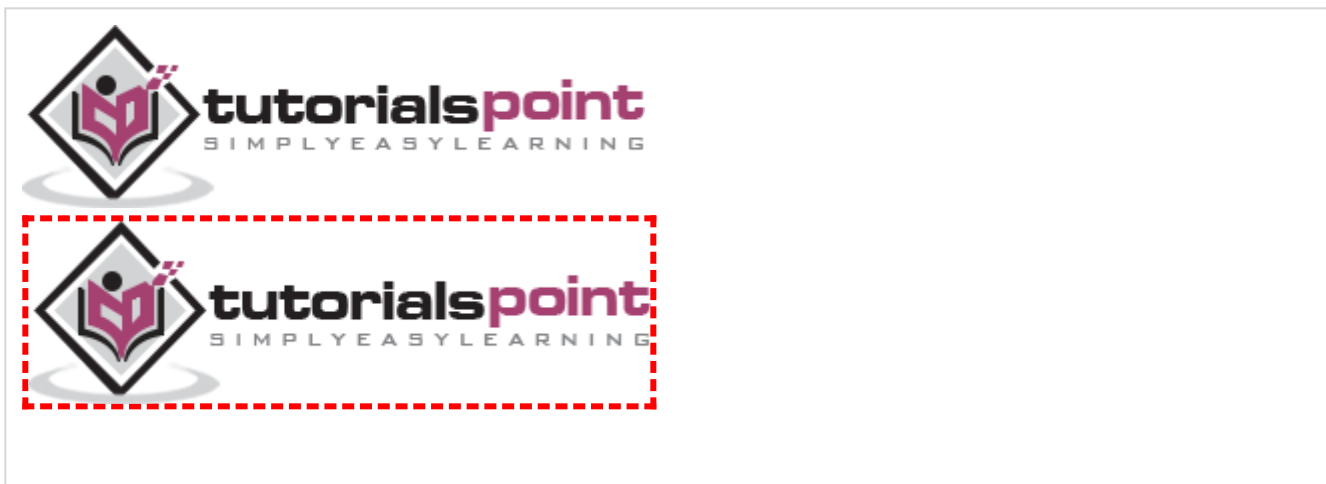
Here is the example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <img style = "border:0px;" src = "/css/images/logo.png" />
    <br />
    <img style = "border:3px dashed red;" src = "/css/images/logo.png" />
  </body>
</html>
```

It will produce the following result –



The Image Height Property

The *height* property of an image is used to set the height of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <img style = "border:1px solid red; height:100px;" src = "/css/images/logo.png" />
  </body>
</html>
```



</html>

It will produce the following result –



The Image Width Property

The *width* property of an image is used to set the width of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <img style = "border:1px solid red; width:150px;" src = "/css/images/logo.png" />
    <br />
    <img style = "border:1px solid red; width:100%;" src = "/css/images/logo.png" />
  </body>
</html>
```

It will produce the following result –



The -moz-opacity Property

The *-moz-opacity* property of an image is used to set the opacity of an image. This property is used to create a transparent image in Mozilla. IE uses **filter:alpha(opacity=x)** to create transparent images.

In Mozilla (-moz-opacity:x) x can be a value from 0.0 - 1.0. A lower value makes the element more transparent (The same things goes for the CSS3-valid syntax opacity:x).

In IE (filter:alpha(opacity=x)) x can be a value from 0 - 100. A lower value makes the element more transparent.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <img style = "border:1px solid red; -moz-opacity:0.4; filter:alpha(opacity=40);" src =
  </body>
</html>
```

It will produce the following result –





```
<style type = "text/css">
  a:link {color: #000000}
  a:visited {color: #006600}
  a:hover {color: #FFCC00}
  a:active {color: #FF00CC}
</style>
```

Now, we will see how to use these properties to give different effects to hyperlinks.

Set the Color of Links

The following example demonstrates how to set the link color. Possible values could be any color name in any valid format.

```
<html>
  <head>
    <style type = "text/css">
      a:link {color:#000000}
    </style>
  </head>

  <body>
    <a href = "">Link</a>
  </body>
</html>
```

[Live Demo](#)

It will produce the following black link –

[Link](#)

Set the Color of Visited Links

The following example demonstrates how to set the color of visited links. Possible values could be any color name in any valid format.

```
<html>
  <head>
    <style type = "text/css">
      a:visited {color: #006600}
    </style>
  </head>
```

[Live Demo](#)



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

It will produce the following link. Once you will click this link, it will change its color to green.

[link](#)

Change the Color of Links when Mouse is Over

The following example demonstrates how to change the color of links when we bring a mouse pointer over that link. Possible values could be any color name in any valid format.

[Live Demo](#)

```
<html>  
  <head>  
    <style type = "text/css">  
      a:hover {color: #FFCC00}  
    </style>  
  </head>  
  
  <body>  
    <a href = "">Link</a>  
  </body>  
</html>
```

It will produce the following link. Now, you bring your mouse over this link and you will see that it changes its color to yellow.

[Link](#)

Change the Color of Active Links

The following example demonstrates how to change the color of active links. Possible values could be any color name in any valid format.

[Live Demo](#)

```
<html>  
  <head>  
    <style type = "text/css">  
      a:active {color: #FF00CC}  
    </style>  
  </head>  
  
  <body>
```



It will produce the following link. It will change its color to pink when the user clicks it.

[Link](#)

CSS - Tables

Now, we will see how to use these properties with examples.

The border-collapse Property

This property can have two values *collapse* and *separate*. The following example uses both the values –

Live Demo

```
<html>
<head>
  <style type = "text/css">
    table.one {border-collapse:collapse;}
    table.two {border-collapse:separate;}

    td.a {
      border-style:dotted;
      border-width:3px;
      border-color:#000000;
      padding: 10px;
    }
    td.b {
      border-style:solid;
      border-width:3px;
      border-color:#333333;
      padding:10px;
    }
  </style>
</head>

<body>
  <table class = "one">
    <caption>Collapse Border Example</caption>
    <tr><td class = "a"> Cell A Collapse Example</td></tr>
    <tr><td class = "b"> Cell B Collapse Example</td></tr>
  </table>
  <br />
```

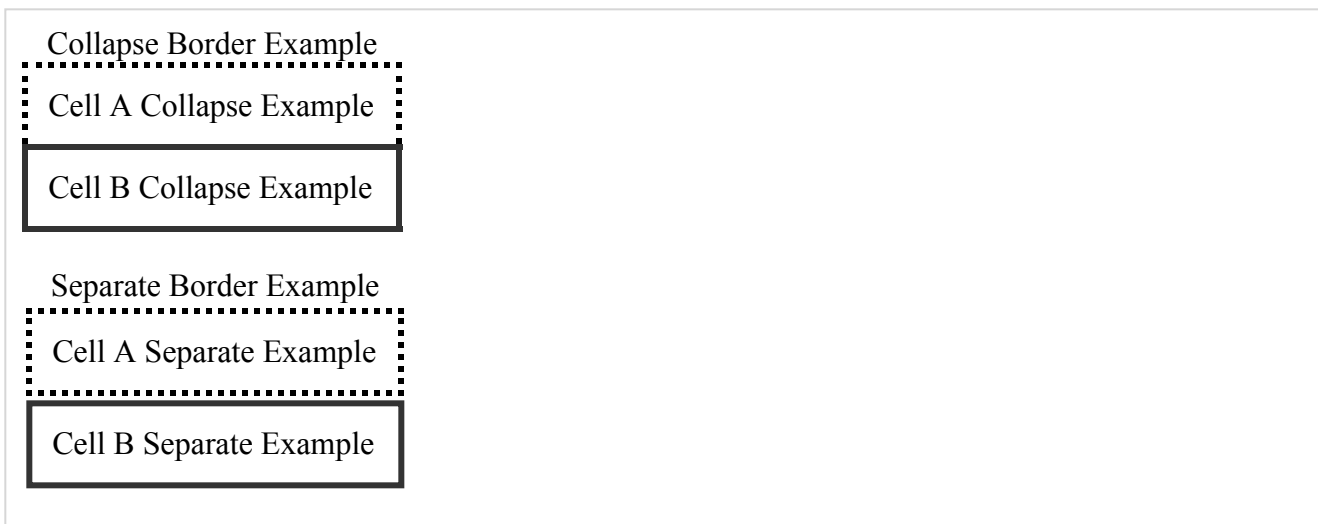


```

<tr><td class = "b"> Cell B Collapse Example</td></tr>
</table>
</body>
</html>

```

It will produce the following result –



The border-spacing Property

The border-spacing property specifies the distance that separates adjacent cells' borders. It can take either one or two values; these should be units of length.

If you provide one value, it will apply to both vertical and horizontal borders. Or you can specify two values, in which case, the first refers to the horizontal spacing and the second to the vertical spacing –

NOTE – Unfortunately, this property does not work in Netscape 7 or IE 6.

```

<style type="text/css">
  /* If you provide one value */
  table.example {border-spacing:10px;}
  /* This is how you can provide two values */
  table.example {border-spacing:10px; 15px;}
</style>

```

Now let's modify the previous example and see the effect –

```

<html>
<head>
  <style type = "text/css">
    table.one {
      border-collapse:separate;

```

Live Demo



```

table.two {
    border-collapse: separate;
    width: 400px;
    border-spacing: 10px 50px;
}
</style>
</head>

<body>

<table class = "one" border = "1">
    <caption>Separate Border Example with border-spacing</caption>
    <tr><td> Cell A Collapse Example</td></tr>
    <tr><td> Cell B Collapse Example</td></tr>
</table>
<br />

<table class = "two" border = "1">
    <caption>Separate Border Example with border-spacing</caption>
    <tr><td> Cell A Separate Example</td></tr>
    <tr><td> Cell B Separate Example</td></tr>
</table>

</body>
</html>

```

It will produce the following result –

Separate Border Example with border-spacing

Cell A Collapse Example

Cell B Collapse Example

Separate Border Example with border-spacing

Cell A Separate Example

Cell B Separate Example



placed in relationship to the table. The table that follows lists the possible values.

This property can have one of the four values *top*, *bottom*, *left* or *right*. The following example uses each value.

NOTE – These properties may not work with your IE Browser.

[Live Demo](#)

```
<html>
  <head>
    <style type = "text/css">
      caption.top {caption-side:top}
      caption.bottom {caption-side:bottom}
      caption.left {caption-side:left}
      caption.right {caption-side:right}
    </style>
  </head>

  <body>

    <table style = "width:400px; border:1px solid black;">
      <caption class = "top">
        This caption will appear at the top
      </caption>
      <tr><td > Cell A</td></tr>
      <tr><td > Cell B</td></tr>
    </table>
    <br />

    <table style = "width:400px; border:1px solid black;">
      <caption class = "bottom">
        This caption will appear at the bottom
      </caption>
      <tr><td > Cell A</td></tr>
      <tr><td > Cell B</td></tr>
    </table>
    <br />

    <table style = "width:400px; border:1px solid black;">
      <caption class = "left">
        This caption will appear at the left
      </caption>
      <tr><td > Cell A</td></tr>
      <tr><td > Cell B</td></tr>
    </table>
    <br />

    <table style = "width:400px; border:1px solid black;">
      <caption class = "right">
```



```

<tr><td> Cell B</td></tr>
</table>

</body>
</html>

```

It will produce the following result –

This caption will appear at the top

Cell A
Cell B

This caption will appear at the bottom

Cell A
Cell B

This caption will appear at the left

Cell A
Cell B

This caption will appear at the right

Cell A
Cell B

The empty-cells Property

The empty-cells property indicates whether a cell without any content should have a border displayed.

This property can have one of the three values - *show*, *hide* or *inherit*.

Here is the empty-cells property used to hide borders of empty cells in the <table> element.

Live Demo

```

<html>
  <head>
    <style type = "text/css">
      table.empty {
        width:350px;
        border-collapse:separate;
        empty-cells:hide;
      }
      td.empty {
        padding:5px;

```



```

</style>
</head>

<body>

  <table class = "empty">
    <tr>
      <th></th>
      <th>Title one</th>
      <th>Title two</th>
    </tr>

    <tr>
      <th>Row Title</th>
      <td class = "empty">value</td>
      <td class = "empty">value</td>
    </tr>

    <tr>
      <th>Row Title</th>
      <td class = "empty">value</td>
      <td class = "empty"></td>
    </tr>
  </table>

</body>
</html>

```

It will produce the following result –

	Title one	Title two
Row Title	value	value
Row Title	value	

The table-layout Property

The table-layout property is supposed to help you control how a browser should render or lay out a table.

This property can have one of the three values: *fixed*, *auto* or *inherit*.

The following example shows the difference between these properties.

NOTE – This property is not supported by many browsers so do not rely on this property.



10000000000000000	10000000	100
1000000000000	10000000	100

The border-color property allows you to change the color of the border surrounding an element. You can individually change the color of the bottom, left, top and right sides of an element's border using the properties –



border-left-color changes the color of left border.

border-right-color changes the color of right border.

The following example shows the effect of all these properties –

[Live Demo](#)

```
<html>
  <head>
    <style type = "text/css">
      p.example1 {
        border:1px solid;
        border-bottom-color:#009900; /* Green */
        border-top-color:#FF0000;    /* Red */
        border-left-color:#330000;   /* Black */
        border-right-color:#0000CC;  /* Blue */
      }
      p.example2 {
        border:1px solid;
        border-color:#009900;        /* Green */
      }
    </style>
  </head>

  <body>
    <p class = "example1">
      This example is showing all borders in different colors.
    </p>

    <p class = "example2">
      This example is showing all borders in green color only.
    </p>
  </body>
</html>
```

It will produce the following result –

This example is showing all borders in different colors.

This example is showing all borders in green color only.

The border-style Property

The border-style property allows you to select one of the following styles of border –

none – No border. (Equivalent of border-width:0;)



dashed – Border is a series of short lines.

double – Border is two solid lines.

groove – Border looks as though it is carved into the page.

ridge – Border looks the opposite of groove.

inset – Border makes the box look like it is embedded in the page.

outset – Border makes the box look like it is coming out of the canvas.

hidden – Same as none, except in terms of border-conflict resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using the following properties –

border-bottom-style changes the style of bottom border.

border-top-style changes the style of top border.

border-left-style changes the style of left border.

border-right-style changes the style of right border.

The following example shows all these border styles –

[Live Demo](#)

```
<html>
<head>
</head>

<body>
  <p style = "border-width:4px; border-style:none;">
    This is a border with none width.
  </p>

  <p style = "border-width:4px; border-style:solid;">
    This is a solid border.
  </p>

  <p style = "border-width:4px; border-style:dashed;">
    This is a dashed border.
  </p>

  <p style = "border-width:4px; border-style:double;">
    This is a double border.
  </p>

  <p style = "border-width:4px; border-style:groove;">
    This is a groove border.
  </p>
```



```

<p style = "border-width:4px; border-style:inset;">
    This is a inset border.
</p>

<p style = "border-width:4px; border-style:outset;">
    This is a outset border.
</p>

<p style = "border-width:4px; border-style:hidden;">
    This is a hidden border.
</p>

<p style = "border-width:4px;
border-top-style:solid;
border-bottom-style:dashed;
border-left-style:groove;
border-right-style:double;">
    This is a a border with four different styles.
</p>
</body>
</html>

```

It will produce the following result –

This is a border with none width.

This is a solid border.

This is a dahsed border.

This is a double border.

This is a groove border.

This is aridge border.

This is a inset border.

This is a outset border.

This is a hidden border.

This is a a border with four different styles.



property could be either a length in px, pt or cm or it should be set to *thin*, *medium* or *thick*.

You can individually change the width of the bottom, top, left, and right borders of an element using the following properties –

border-bottom-width changes the width of bottom border.

border-top-width changes the width of top border.

border-left-width changes the width of left border.

border-right-width changes the width of right border.

The following example shows all these border width –

[Live Demo](#)

```
<html>
<head>
</head>

<body>
  <p style = "border-width:4px; border-style:solid;">
    This is a solid border whose width is 4px.
  </p>

  <p style = "border-width:4pt; border-style:solid;">
    This is a solid border whose width is 4pt.
  </p>

  <p style = "border-width:thin; border-style:solid;">
    This is a solid border whose width is thin.
  </p>

  <p style = "border-width:medium; border-style:solid;">
    This is a solid border whose width is medium;
  </p>

  <p style = "border-width:thick; border-style:solid;">
    This is a solid border whose width is thick.
  </p>

  <p style = "border-bottom-width:4px;border-top-width:10px;
    border-left-width: 2px;border-right-width:15px;border-style:solid;">
    This is a a border with four different width.
  </p>
</body>
</html>
```

It will produce the following result –



This is a solid border whose width is 1pt.

This is a solid border whose width is thin.

This is a solid border whose width is medium;

This is a solid border whose width is thick.

This is a a border with four different width.

Border Properties Using Shorthand

The border property allows you to specify color, style, and width of lines in one property –

The following example shows how to use all the three properties into a single property. This is the most frequently used property to set border around any element.

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "border:4px solid red;">
      This example is showing shorthand property for border.
    </p>
  </body>
</html>
```

It will produce the following result –

This example is showing shorthand property for border.

CSS - Margins

- The **margin-right** specifies the right margin of an element.

Now, we will see how to use these properties with examples.

The Margin Property



Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p style = "margin: 15px; border:1px solid black;">
      all four margins will be 15px
    </p>

    <p style = "margin:10px 2%; border:1px solid black;">
      top and bottom margin will be 10px, left and right margin will be 2%
      of the total width of the document.
    </p>

    <p style = "margin: 10px 2% -10px; border:1px solid black;">
      top margin will be 10px, left and right margin will be 2% of the
      total width of the document, bottom margin will be -10px
    </p>

    <p style = "margin: 10px 2% -10px auto; border:1px solid black;">
      top margin will be 10px, right margin will be 2% of the total
      width of the document, bottom margin will be -10px, left margin
      will be set by the browser
    </p>
  </body>
</html>
```

It will produce the following result –

all four margins will be 15px

top and bottom margin will be 10px, left and right margin will be 2% of the total width of the document.

top margin will be 10px, left and right margin will be 2% of the total width of the document, bottom margin will be -10px

top margin will be 10px, right margin will be 2% of the total width of the document, bottom margin will be -10px, left margin will be set by the browser



length, % or auto.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "margin-bottom: 15px; border:1px solid black;">
      This is a paragraph with a specified bottom margin
    </p>

    <p style = "margin-bottom: 5%; border:1px solid black;">
      This is another paragraph with a specified bottom margin in percent
    </p>
  </body>
</html>
```

It will produce the following result –

This is a paragraph with a specified bottom margin

This is another paragraph with a specified bottom margin in percent

The margin-top Property

The margin-top property allows you set top margin of an element. It can have a value in length, % or auto.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "margin-top: 15px; border:1px solid black;">
      This is a paragraph with a specified top margin
    </p>

    <p style = "margin-top: 5%; border:1px solid black;">
      This is another paragraph with a specified top margin in percent
    </p>
  </body>
</html>
```



It will produce the following result –

This is a paragraph with a specified top margin

This is another paragraph with a specified top margin in percent

The margin-left Property

The margin-left property allows you set left margin of an element. It can have a value in length, % or auto.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "margin-left: 15px; border:1px solid black;">
      This is a paragraph with a specified left margin
    </p>

    <p style = "margin-left: 5%; border:1px solid black;">
      This is another paragraph with a specified top margin in percent
    </p>
  </body>
</html>
```

It will produce the following result –

This is a paragraph with a specified left margin

This is another paragraph with a specified top margin in percent

The margin-right Property

The margin-right property allows you set right margin of an element. It can have a value in length, % or auto.

Here is an example –



```
</head>
```

```
<body>
```

```
<p style = "margin-right: 15px; border:1px solid black;">
```

```
    This is a paragraph with a specified right margin
```

```
</p>
```

```
<p style = "margin-right: 5%; border:1px solid black;">
```

```
    This is another paragraph with a specified right margin in percent
```

```
</p>
```

```
</body>
```

```
</html>
```

It will produce the following result –

This is a paragraph with a specified right margin

This is another paragraph with a specified right margin in percent

CSS - Lists

Now, we will see how to use these properties with examples.

The list-style-type Property

The *list-style-type* property allows you to control the shape or style of bullet point (also known as a marker) in the case of unordered lists and the style of numbering characters in ordered lists.

Here are the values which can be used for an unordered list –

Sr.No.	Value & Description
1	none NA
2	disc (default) A filled-in circle
3	circle An empty circle



Here are the values, which can be used for an ordered list –

Value	Description	Example
decimal	Number	1,2,3,4,5
decimal-leading-zero	0 before the number	01, 02, 03, 04, 05
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V
lower-greek	The marker is lower-greek	alpha, beta, gamma
lower-latin	The marker is lower-latin	a, b, c, d, e
upper-latin	The marker is upper-latin	A, B, C, D, E
hebrew	The marker is traditional Hebrew numbering	
armenian	The marker is traditional Armenian numbering	
georgian	The marker is traditional Georgian numbering	
cjk-ideographic	The marker is plain ideographic numbers	
hiragana	The marker is hiragana	a, i, u, e, o, ka, ki
katakana	The marker is katakana	A, I, U, E, O, KA, KI
hiragana-iroha	The marker is hiragana-iroha	i, ro, ha, ni, ho, he, to
katakana-iroha	The marker is katakana-iroha	I, RO, HA, NI, HO, HE, TO

Here is an example –

```
<html>
  <head>
    </head>
```

Live Demo



```
</li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ul style = "list-style-type:square;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ul>

<ol style = "list-style-type:decimal;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>

<ol style = "list-style-type:lower-alpha;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>

<ol style = "list-style-type:lower-roman;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>
</body>
</html>
```

It will produce the following result –



- Maths
- Social Science
- Physics

1. Maths
2. Social Science
3. Physics

- a. Maths
- b. Social Science
- c. Physics

- i. Maths
- ii. Social Science
- iii. Physics

The list-style-position Property

The *list-style-position* property indicates whether the marker should appear inside or outside of the box containing the bullet points. It can have one the two values –

Sr.No.	Value & Description
1	none NA
2	inside If the text goes onto a second line, the text will wrap underneath the marker. It will also appear indented to where the text would have started if the list had a value of outside.
3	outside If the text goes onto a second line, the text will be aligned with the start of the first line (to the right of the bullet).

Here is an example –

```
<html>
<head>
</head>
```

Live Demo



```

</li>Social Science</li>
<li>Physics</li>
</ul>

<ul style = "list-style-type:square;list-style-position:inside;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ul>

<ol style = "list-style-type:decimal;list-stlye-position:outside;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>

<ol style = "list-style-type:lower-alpha;list-style-position:inside;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>
</body>
</html>

```

It will produce the following result –

- Maths
 - Social Science
 - Physics
-
- Maths
 - Social Science
 - Physics
-
1. Maths
 2. Social Science
 3. Physics
-
- a. Maths
 - b. Social Science
 - c. Physics

The list-style-image Property

The *list-style-image* allows you to specify an image so that you can use your own bullet style. The syntax is similar to the background-image property with the letters url starting the value of the



Live Demo

```
<html>
  <head>
  </head>

  <body>
    <ul>
      <li style = "list-style-image: url(/images/bullet.gif);">Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ul>

    <ol>
      <li style = "list-style-image: url(/images/bullet.gif);">Maths</li>
      <li>Social Science</li>
      <li>Physics</li>
    </ol>
  </body>
</html>
```

It will produce the following result –

- Maths
 - Social Science
 - Physics
-
- Maths
 - 2. Social Science
 - 3. Physics

The list-style Property

The *list-style* allows you to specify all the list properties into a single expression. These properties can appear in any order.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <ul style = "list-style: inside square;">
```



```
</ul>

<ol style = "list-style: outside upper-alpha;">
  <li>Maths</li>
  <li>Social Science</li>
  <li>Physics</li>
</ol>
</body>
</html>
```

It will produce the following result –

- Maths
 - Social Science
 - Physics
- A. Maths
B. Social Science
C. Physics

The marker-offset Property

The *marker-offset* property allows you to specify the distance between the marker and the text relating to that marker. Its value should be a length as shown in the following example –

Unfortunately, this property is not supported in IE 6 or Netscape 7.

Here is an example –

Live Demo

```
<html>
<head>
</head>

<body>
  <ul style = "list-style: inside square; marker-offset:2em;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ul>

  <ol style = "list-style: outside upper-alpha; marker-offset:2cm;">
    <li>Maths</li>
    <li>Social Science</li>
    <li>Physics</li>
  </ol>
```



It will produce the following result –

- Maths
- Social Science
- Physics

A. Maths
B. Social Science
C. Physics

CSS - Paddings

•

The **padding** serves as shorthand for the preceding properties.

Now, we will see how to use these properties with examples.

The padding-bottom Property

The *padding-bottom* property sets the bottom padding (space) of an element. This can take a value in terms of length of %.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p style = "padding-bottom: 15px; border:1px solid black;">
      This is a paragraph with a specified bottom padding
    </p>

    <p style = "padding-bottom: 5%; border:1px solid black;">
      This is another paragraph with a specified bottom padding in percent
    </p>
  </body>
</html>
```

It will produce the following result –



This is another paragraph with a specified bottom padding in percent

The padding-top Property

The *padding-top* property sets the top padding (space) of an element. This can take a value in terms of length of %.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "padding-top: 15px; border:1px solid black;">
      This is a paragraph with a specified top padding
    </p>

    <p style = "padding-top: 5%; border:1px solid black;">
      This is another paragraph with a specified top padding in percent
    </p>
  </body>
</html>
```

It will produce the following result –

This is a paragraph with a specified top padding

This is another paragraph with a specified top padding in percent

The padding-left Property

The *padding-left* property sets the left padding (space) of an element. This can take a value in terms of length of %.

Here is an example –

[Live Demo](#)



```
<body>
  <p style = "padding-left: 15px; border:1px solid black;">
    This is a paragraph with a specified left padding
  </p>

  <p style = "padding-left: 15%; border:1px solid black;">
    This is another paragraph with a specified left padding in percent
  </p>
</body>
</html>
```

It will produce the following result –

This is a paragraph with a specified left padding

This is another paragraph with a specified left padding in percent

The padding-right Property

The *padding-right* property sets the right padding (space) of an element. This can take a value in terms of length or %.

Here is an example –

```
<html>
  <head>
  </head>

  <body>
    <p style = "padding-right: 15px; border:1px solid black;">
      This is a paragraph with a specified right padding
    </p>

    <p style = "padding-right: 5%; border:1px solid black;">
      This is another paragraph with a specified right padding in percent
    </p>
  </body>
</html>
```

Live Demo

It will produce the following result –



The Padding Property

The *padding* property sets the left, right, top and bottom padding (space) of an element. This can take a value in terms of length of %.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "padding: 15px; border:1px solid black;">
      all four padding will be 15px
    </p>

    <p style = "padding:10px 2%; border:1px solid black;">
      top and bottom padding will be 10px, left and right
      padding will be 2% of the total width of the document.
    </p>

    <p style = "padding: 10px 2% 10px; border:1px solid black;">
      top padding will be 10px, left and right padding will
      be 2% of the total width of the document, bottom padding will be 10px
    </p>

    <p style = "padding: 10px 2% 10px 10px; border:1px solid black;">
      top padding will be 10px, right padding will be 2% of
      the total width of the document, bottom padding and top padding will be 10px
    </p>
  </body>
</html>
```

It will produce the following result –



top and bottom padding will be 10px, left and right padding will be 2% of the total width of the document.

top padding will be 10px, left and right padding will be 2% of the total width of the document, bottom padding will be 10px

top padding will be 10px, right padding will be 2% of the total width of the document, bottom padding and top padding will be 10px

CSS - Cursors

1

auto

Shape of the cursor depends on the context area it is over. For example an I over text, a hand over a link, and so on...

2

crosshair

A crosshair or plus sign

3

default

An arrow

4

pointer

A pointing hand (in IE 4 this value is hand)

5

move

The I bar

6

e-resize

The cursor indicates that an edge of a box is to be moved right (east)

7



8

nw-resize

The cursor indicates that an edge of a box is to be moved up and left (north/west)

9

n-resize

The cursor indicates that an edge of a box is to be moved up (north)

10

se-resize

The cursor indicates that an edge of a box is to be moved down and right (south/east)

11

sw-resize

The cursor indicates that an edge of a box is to be moved down and left (south/west)

12

s-resize

The cursor indicates that an edge of a box is to be moved down (south)

13

w-resize

The cursor indicates that an edge of a box is to be moved left (west)

14

text

The I bar

15

wait

An hour glass

16

help

A question mark or balloon, ideal for use over help buttons

17

<url>

The source of a cursor image file

NOTE – You should try to use only these values to add helpful information for users, and in places, they would expect to see that cursor. For example, using the crosshair when someone hovers over a



Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p>Move the mouse over the words to see the cursor change:</p>

    <div style = "cursor:auto">Auto</div>
    <div style = "cursor:crosshair">Crosshair</div>
    <div style = "cursor:default">Default</div>

    <div style = "cursor:pointer">Pointer</div>
    <div style = "cursor:move">Move</div>
    <div style = "cursor:e-resize">e-resize</div>
    <div style = "cursor:ne-resize">ne-resize</div>
    <div style = "cursor:nw-resize">nw-resize</div>

    <div style = "cursor:n-resize">n-resize</div>
    <div style = "cursor:se-resize">se-resize</div>
    <div style = "cursor:sw-resize">sw-resize</div>
    <div style = "cursor:s-resize">s-resize</div>
    <div style = "cursor:w-resize">w-resize</div>

    <div style = "cursor:text">text</div>
    <div style = "cursor:wait">wait</div>
    <div style = "cursor:help">help</div>
  </body>
</html>
```

It will produce the following result –



Crosshair
Default
Pointer
Move
e-resize
ne-resize
nw-resize
n-resize
se-resize
sw-resize
s-resize
w-resize
text
wait
help

CSS - Outlines

The **outline-width** property is used to set the width of the outline.

The **outline-style** property is used to set the line style for the outline.

The **outline-color** property is used to set the color of the outline.

The **outline** property is used to set all the above three properties in a single statement.

The outline-width Property

The *outline-width* property specifies the width of the outline to be added to the box. Its value should be a length or one of the values *thin*, *medium*, or *thick*, just like the border-width attribute.

A width of zero pixels means no outline.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

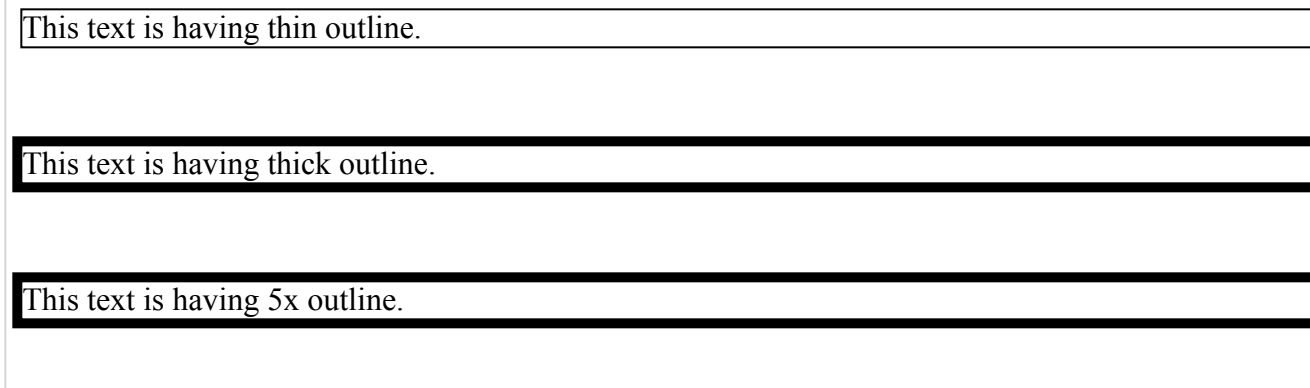
  <body>
    <p style = "outline-width:thin; outline-style:solid;">
      This text is having thin outline.
    </p>
    <br />

    <p style = "outline-width:thick; outline-style:solid;">
      This text is having thick outline.
```



```
<p style = "outline-width:5px, outline-style:solid, >  
    This text is having 5x outline.  
</p>  
</body>  
</html>
```

It will produce the following result –



The outline-style Property

The *outline-style* property specifies the style for the line (solid, dotted, or dashed) that goes around an element. It can take one of the following values –

- none** – No border. (Equivalent of outline-width:0;)
- solid** – Outline is a single solid line.
- dotted** – Outline is a series of dots.
- dashed** – Outline is a series of short lines.
- double** – Outline is two solid lines.
- groove** – Outline looks as though it is carved into the page.
- ridge** – Outline looks the opposite of groove.
- inset** – Outline makes the box look like it is embedded in the page.
- outset** – Outline makes the box look like it is coming out of the canvas.
- hidden** – Same as none.

Here is an example –

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

Live Demo



```

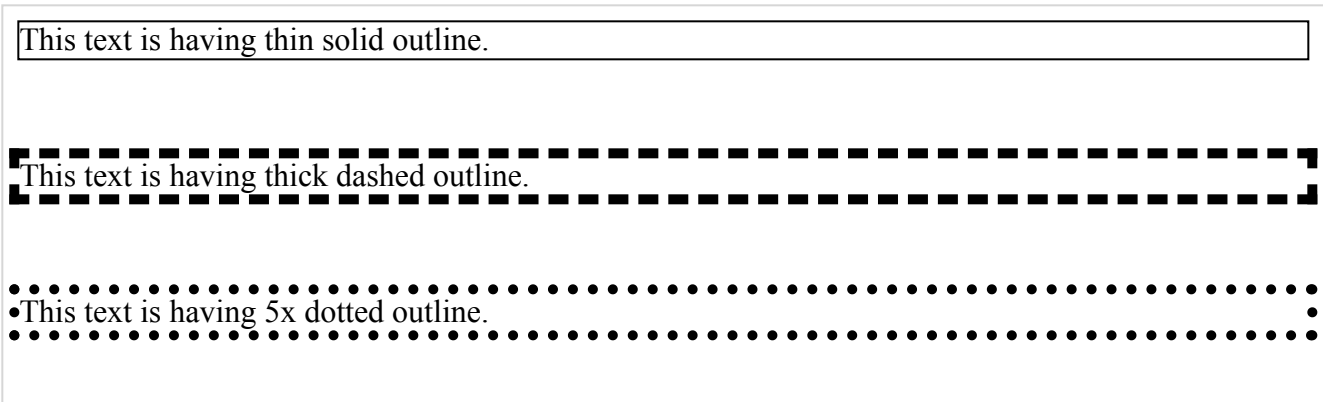
<br />

<p style = "outline-width:thick; outline-style:dashed;">
    This text is having thick dashed outline.
</p>
<br />

<p style = "outline-width:5px;outline-style:dotted;">
    This text is having 5x dotted outline.
</p>
</body>
</html>

```

It will produce the following result –



The outline-color Property

The *outline-color* property allows you to specify the color of the outline. Its value should either be a color name, a hex color, or an RGB value, as with the color and border-color properties.

Here is an example –

```

<html>
  <head>
  </head>

  <body>
    <p style = "outline-width:thin; outline-style:solid;outline-color:red">
      This text is having thin solid red  outline.
    </p>
    <br />

    <p style = "outline-width:thick; outline-style:dashed;outline-color:#009900">
      This text is having thick dashed green outline.
    </p>

```

Live Demo



THIS text is having 5x dotted blue outline.

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

It will produce the following result –

This text is having thin solid red outline.

This text is having thick dashed green outline.

This text is having 5x dotted blue outline.

The outline Property

The *outline* property is a shorthand property that allows you to specify values for any of the three properties discussed previously in any order but in a single statement.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p style = "outline:thin solid red;">
      This text is having thin solid red outline.
    </p>
    <br />

    <p style = "outline:thick dashed #009900;">
      This text is having thick dashed green outline.
    </p>
    <br />

    <p style = "outline:5px dotted rgb(13,33,232);">
      This text is having 5x dotted blue outline.
    </p>
  </body>
</html>
```



This text is having thick dashed green outline.

This text is having 5x dotted blue outline.

CSS - Dimension

- The **max-width** property is used to set the maximum width that a box can be.
- The **min-width** property is used to set the minimum width that a box can be.

The Height and Width Properties

The *height* and *width* properties allow you to set the height and width for boxes. They can take values of a length, a percentage, or the keyword auto.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "width:400px; height:100px; border:1px solid red; padding:5px; margin:10px;">
      This paragraph is 400pixels wide and 100 pixels high
    </p>
  </body>
</html>
```

It will produce the following result –



The line-height Property

The *line-height* property allows you to increase the space between lines of text. The value of the line-height property can be a number, a length, or a percentage.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p style = "width:400px; height:100px; border:1px solid red; padding:5px; margin:10px;">
      This paragraph is 400pixels wide and 100 pixels high and here line height is 30pixels
      This paragraph is 400 pixels wide and 100 pixels high and here line height is 30pixels
    </p>
  </body>
</html>
```

It will produce the following result –

This paragraph is 400pixels wide and 100 pixels high and here line height is 30pixels. This paragraph is 400 pixels wide and 100 pixels high and here line height is 30pixels.

The max-height Property

The *max-height* property allows you to specify maximum height of a box. The value of the max-height property can be a number, a length, or a percentage.

NOTE – This property does not work in either Netscape 7 or IE 6.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
```



This paragraph is 400px wide and max height is 10px

This paragraph is 400px wide and max height is 10px

This paragraph is 400px wide and max height is 10px

This paragraph is 400px wide and max height is 10px

</p>

</body>

</html>

It will produce the following result –

This paragraph is 400px wide and max height is 10px This
 paragraph is 400px wide and max height is 10px This
 paragraph is 400px wide and max height is 10px This
 paragraph is 400px wide and max height is 10px



The min-height Property

The *min-height* property allows you to specify minimum height of a box. The value of the min-height property can be a number, a length, or a percentage.

NOTE – This property does not work in either Netscape 7 or IE 6.

Here is an example –

```
<html>
<head>
</head>

<body>
  <p style = "width:400px; min-height:200px; border:1px solid red; padding:5px; margin:10px 0 0 0;">
    This paragraph is 400px wide and min height is 200px
    This paragraph is 400px wide and min height is 200px
    This paragraph is 400px wide and min height is 200px
    This paragraph is 400px wide and min height is 200px
  </p>
```

Live Demo



It will produce the following result –

This paragraph is 400px wide and min height is 200px This
paragraph is 400px wide and min height is 200px This
paragraph is 400px wide and min height is 200px This
paragraph is 400px wide and min height is 200px



The max-width Property

The *max-width* property allows you to specify maximum width of a box. The value of the max-width property can be a number, a length, or a percentage.

NOTE – This property does not work in either Netscape 7 or IE 6.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p style = "max-width:100px; height:200px; border:1px solid red; padding:5px; margin:10px 0 0 0;">
      This paragraph is 200px high and max width is 100px
      This paragraph is 200px high and max width is 100px
      This paragraph is 200px high and max width is 100px
      This paragraph is 200px high and max width is 100px
      This paragraph is 200px high and max width is 100px
    </p>
    <img alt = "logo" src = "/images/css.gif" width = "95" height = "84" />
  </body>
</html>
```



is 200px high
and max width
is 100px This
paragraph is
200px high and
max width is
100px This
paragraph is
200px high and
max width is
100px This

W3C® and
CSS html
Web Development
paragraph is
200px high and
max width is
100px

The min-width Property

The *min-width* property allows you to specify minimum width of a box. The value of the min-width property can be a number, a length, or a percentage.

NOTE – This property does not work in either Netscape 7 or IE 6.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p style = "min-width:400px; height:100px; border:1px solid red; padding:5px; margin:10px 0 0 0;">
      This paragraph is 100px high and min width is 400px
      This paragraph is 100px high and min width is 400px
    </p>
    <img alt = "logo" src = "/css/images/css.gif" width = "95" height = "84" />
  </body>
</html>
```

It will produce the following result –



```

display: block;
border: 1px solid red;
padding: 5px;
margin-top: 5px;
width: 300px;
height: 50px;
overflow: auto;
}
</style>
</head>

<body>
<p>Example of scroll value:</p>
<div class = "scroll">
    I am going to keep lot of content here just to show you how
    scrollbars works if there is an overflow in an element box.
    This provides your horizontal as well as vertical scrollbars.
</div>
<br />

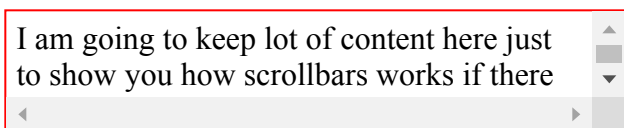
<p>Example of auto value:</p>

<div class = "auto">
    I am going to keep lot of content here just to show you how
    scrollbars works if there is an overflow in an element box.
    This provides your horizontal as well as vertical scrollbars.
</div>
</body>
</html>

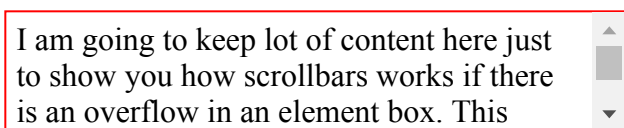
```

It will produce the following result –

Example of scroll value:



Example of auto value:





The box and its contents are shown to the user.

2

hidden

The box and its content are made invisible, although they still affect the layout of the page.

3

collapse

This is for use only with dynamic table columns and row effects.

Here is an example –

Live Demo

```
<html>
  <head>
  </head>

  <body>
    <p>
      This paragraph should be visible in normal way.
    </p>

    <p style = "visibility:hidden;">
      This paragraph should not be visible.
    </p>
  </body>
</html>
```

It will produce the following result –

This paragraph should be visible in normal way.

CSS - Positioning

- Move Up - Use a negative value for *top*.
- Move Down - Use a positive value for *top*.

NOTE – You can use *bottom* or *right* values as well in the same way as *top* and *left*.

Here is the example –

Live Demo



```
<body>
  <div style = "position:relative; left:80px; top:2px; background-color:yellow;">
    This div has relative positioning.
  </div>
</body>
</html>
```

It will produce the following result –

This div has relative positioning.

Absolute Positioning

An element with **position: absolute** is positioned at the specified coordinates relative to your screen top-left corner.

You can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

Move Left - Use a negative value for *left*.

Move Right - Use a positive value for *left*.

Move Up - Use a negative value for *top*.

Move Down - Use a positive value for *top*.

NOTE – You can use *bottom* or *right* values as well in the same way as top and left.

Here is an example –

```
<html>
  <head>
  </head>

  <body>
    <div style = "position:absolute; left:80px; top:20px; background-color:yellow;">
      This div has absolute positioning.
    </div>
  </body>
</html>
```

Live Demo



Fixed Positioning

Fixed positioning allows you to fix the position of an element to a particular spot on the page, regardless of scrolling. Specified coordinates will be relative to the browser window.

You can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

Move Left - Use a negative value for *left*.

Move Right - Use a positive value for *left*.

Move Up - Use a negative value for *top*.

Move Down - Use a positive value for *top*.

NOTE – You can use *bottom* or *right* values as well in the same way as *top* and *left*.

Here is an example –

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <div style = "position:fixed; left:80px; top:20px; background-color:yellow;">
      This div has fixed positioning.
    </div>
  </body>
</html>
```

This div has fixed positioning.

CSS - Layers

```
<body> <div style = "background-color:red; width:300px; height:100px; position:relative; top:10px; left:80px; z-index:2"> </div> <div style = "background-color:yellow; width:300px; height:100px; position:relative; top:-60px; left:35px; z-index:1;"> </div> <div style = "background-color:green; width:300px; height:100px; position:relative; top:-220px; left:120px; z-index:3;"> </div> </body> </html>
```

It will produce the following result –



CSS - Pseudo Classes

The most commonly used pseudo-classes are as follows –

Sr.No.	Value & Description
1	:link Use this class to add special style to an unvisited link.
2	:visited Use this class to add special style to a visited link.
3	:hover Use this class to add special style to an element when you mouse over it.
4	:active Use this class to add special style to an active element.
5	:focus Use this class to add special style to an element while the element has focus.
6	:first-child



7

:lang

Use this class to specify a language to use in a specified element.

While defining pseudo-classes in a `<style>...</style>` block, following points should be noted –

`a:hover` MUST come after `a:link` and `a:visited` in the CSS definition in order to be effective.

`a:active` MUST come after `a:hover` in the CSS definition in order to be effective.

Pseudo-class names are not case-sensitive.

Pseudo-class are different from CSS classes but they can be combined.

The :link pseudo-class

The following example demonstrates how to use the `:link` class to set the link color. Possible values could be any color name in any valid format.

[Live Demo](#)

```
<html>
  <head>
    <style type = "text/css">
      a:link {color:#000000}
    </style>
  </head>

  <body>
    <a href = "">Black Link</a>
  </body>
</html>
```

It will produce the following black link –

[Black Link](#)

The :visited pseudo-class

The following is the example which demonstrates how to use the `:visited` class to set the color of visited links. Possible values could be any color name in any valid format.

[Live Demo](#)



```
    a:visited {color: #006600}
</style>
</head>

<body>
    <a href = "">Click this link</a>
</body>
</html>
```

This will produce following link. Once you will click this link, it will change its color to green.

[Click this link](#)

The :hover pseudo-class

The following example demonstrates how to use the `:hover` class to change the color of links when we bring a mouse pointer over that link. Possible values could be any color name in any valid format.

Live Demo

```
<html>
<head>
    <style type = "text/css">
        a:hover {color: #FFCC00}
    </style>
</head>

<body>
    <a href = "">Bring Mouse Here</a>
</body>
</html>
```

It will produce the following link. Now you bring your mouse over this link and you will see that it changes its color to yellow.

[Bring Mouse Here](#)

The :active pseudo-class

The following example demonstrates how to use the `:active` class to change the color of active links. Possible values could be any color name in any valid format.



```
<style type = "text/css">
  a:active {color: #FF00CC}
</style>
</head>

<body>
  <a href = "">Click This Link</a>
</body>
</html>
```

It will produce the following link. When a user clicks it, the color changes to pink.

[Click This Link](#)

The :focus pseudo-class

The following example demonstrates how to use the *:focus* class to change the color of focused links. Possible values could be any color name in any valid format.

Live Demo

```
<html>
  <head>
    <style type = "text/css">
      a:focus {color: #0000FF}
    </style>
  </head>

  <body>
    <a href = "">Click this Link</a>
  </body>
</html>
```

It will produce the following link. When this link gets focused, its color changes to orange. The color changes back when it loses focus.

[Click this Link](#)

The :first-child pseudo-class

The *:first-child* pseudo-class matches a specified element that is the first child of another element and adds special style to that element that is the first child of some other element.



Live Demo

```
<html>
  <head>
    <style type = "text/css">
      div > p:first-child {
        text-indent: 25px;
      }
    </style>
  </head>

  <body>

    <div>
      <p>First paragraph in div. This paragraph will be indented</p>
      <p>Second paragraph in div. This paragraph will not be indented</p>
    </div>
    <p>But it will not match the paragraph in this HTML:</p>

    <div>
      <h3>Heading</h3>
      <p>The first paragraph inside the div. This paragraph will not be effected.</p>
    </div>

  </body>
</html>
```

It will produce the following result –

First paragraph in div. This paragraph will be indented

Second paragraph in div. This paragraph will not be indented

But it will not match the paragraph in this HTML:

Heading

The first paragraph inside the div. This paragraph will not be effected.

The :lang pseudo-class

The language pseudo-class *:lang*, allows constructing selectors based on the language setting for specific tags.

This class is useful in documents that must appeal to multiple languages that have different conventions for certain language constructs. For example, the French language typically uses angle brackets (< and >) for quoting purposes, while the English language uses quote marks (' and ').



Live Demo

```
<html>
  <head>
    <style type = "text/css">

      /* Two Levels of quotes for two Languages*/
      :lang(en) { quotes: '""' '""' '""' '""'; }
      :lang(fr) { quotes: "<<" ">>" "<" ">"; }
    </style>
  </head>

  <body>
    <p>...<q lang = "fr">A quote in a paragraph</q>...</p>
  </body>
</html>
```

The :lang selectors will apply to all the elements in the document. However, not all elements make use of the quotes property, so the effect will be transparent for most elements.

It will produce the following result –

...<<A quote in a paragraph>>...

CSS - Pseudo Elements

The most commonly used pseudo-elements are as follows –

Sr.No.	Value & Description
1	:first-line Use this element to add special styles to the first line of the text in a selector.
2	:first-letter Use this element to add special style to the first letter of the text in a selector.
3	:before Use this element to insert some content before an element.



The :first-line pseudo-element

The following example demonstrates how to use the *:first-line* element to add special effects to the first line of elements in the document.

[Live Demo](#)

```
<html>
  <head>
    <style type = "text/css">
      p:first-line { text-decoration: underline; }
      p.noline:first-line { text-decoration: none; }
    </style>
  </head>

  <body>
    <p class = "noline">
      This line would not have any underline because this belongs to nline class.
    </p>

    <p>
      The first line of this paragraph will be underlined as defined in the
      CSS rule above. Rest of the lines in this paragraph will remain normal.
      This example shows how to use :first-line pseduo element to give effect
      to the first line of any HTML element.
    </p>
  </body>
</html>
```

It will produce the following link –

This line would not have any underline because this belongs to nline class.

The first line of this paragraph will be underlined as defined in the CSS rule above. Rest of the lines
in this paragraph will remain normal. This example shows how to use :first-line pseduo element to
give effect to the first line of any HTML element.

The :first-letter pseudo-element

The following example demonstrates how to use the *:first-letter* element to add special effects to the first letter of elements in the document.

[Live Demo](#)

```
<html>
  <head>
```



```

</style>
</head>

<body>
  <p class = "normal">
    First character of this paragraph will be normal and will have font size 10 px;
  </p>

  <p>
    The first character of this paragraph will be 5em big as defined in the
    CSS rule above. Rest of the characters in this paragraph will remain
    normal. This example shows how to use :first-letter pseduo element
    to give effect to the first characters  of any HTML element.
  </p>
</body>
</html>

```

It will produce the following black link –

first character of this paragraph will be normal and will have font size 10 px;

The first character of this paragraph will be 5em big and in red color as defined in the CSS rule above. Rest of the characters in this paragraph will remain normal. This example shows how to use :first-letter pseduo element to give effect to the first characters of any HTML element.

The :before pseudo-element

The following example demonstrates how to use the *:before* element to add some content before any element.

```

<html>
  <head>
    <style type = "text/css">
      p:before {
        content: url(/images/bullet.gif)
      }
    </style>
  </head>

  <body>

```

Live Demo



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

It will produce the following black link –

- This line will be preceded by a bullet.
- This line will be preceded by a bullet.
- This line will be preceded by a bullet.

The :after pseudo-element

The following example demonstrates how to use the *:after* element to add some content after any element.

[Live Demo](#)

```
<html>  
  <head>  
    <style type = "text/css">  
      p:after {  
        content: url(/images/bullet.gif)  
      }  
    </style>  
  </head>  
  
  <body>  
    <p> This line will be succeeded by a bullet.</p>  
    <p> This line will be succeeded by a bullet.</p>  
    <p> This line will be succeeded by a bullet.</p>  
  </body>  
</html>
```

It will produce the following black link –

This line will be succeeded by a bullet.■

This line will be succeeded by a bullet.■

This line will be succeeded by a bullet.■

CSS - @ Rules



start of the style sheet before any of the rules, and its value is a URL.

It can be written in one of the two following ways –

```
<style type = "text/css">
  <!--
    @import "mystyle.css";
    or
    @import url("mystyle.css");
    .....other CSS rules .....
  -->
</style>
```

The significance of the @import rule is that it allows you to develop your style sheets with a modular approach. You can create various style sheets and then include them wherever you need them.

The @charset Rule

If you are writing your document using a character set other than ASCII or ISO-8859-1 you might want to set the @charset rule at the top of your style sheet to indicate what character set the style sheet is written in.

The @charset rule must be written right at the beginning of the style sheet without even a space before it. The value is held in quotes and should be one of the standard character-sets. For example –

```
<style type = "text/css">
  <!--
    @charset "iso-8859-1"
    .....other CSS rules .....
  -->
</style>
```

The @font-face Rule

The @font-face rule is used to exhaustively describe a font face for use in a document. @font-face may also be used to define the location of a font for download, although this may run into implementation-specific limits.

In general, @font-face is extremely complicated, and its use is not recommended for any except those who are expert in font metrics.

Here is an example –

```
<style type = "text/css">
  <!--
    @font-face {
```



```
@font-face {
    font-family: Santiago;
    src: local ("Santiago"),
    url("http://www.font.site/s/santiago.ttf")
    format("truetype");
    unicode-range: U+??,U+100-220;
    font-size: all;
    font-family: sans-serif;
}
-->
</style>
```

The !important Rule

Cascading Style Sheets cascade. It means that the styles are applied in the same order as they are read by the browser. The first style is applied and then the second and so on.

The !important rule provides a way to make your CSS cascade. It also includes the rules that are to be applied always. A rule having a !important property will always be applied, no matter where that rule appears in the CSS document.

For example, in the following style sheet, the paragraph text will be black, even though the first style property applied is red:

```
<style type = "text/css">
    <!--
        p { color: #ff0000; }
        p { color: #000000; }
    -->
</style>
```

So, if you wanted to make sure that a property always applied, you would add the !important property to the tag. So, to make the paragraph text always red, you should write it as follows –

```
<html>
  <head>
    <style type = "text/css">
      p { color: #ff0000 !important; }
      p { color: #000000; }
    </style>
  </head>

  <body>
    <p>Tutorialspoint.com</p>
  </body>
</html>
```

[Live Demo](#)



Tutorialspoint.com

CSS Filters - Text and Image Effects

1

opacity

Level of the opacity. 0 is fully transparent, 100 is fully opaque.

2

finishopacity

Level of the opacity at the other end of the object.

3

style

The shape of the opacity gradient.

0 = uniform

1 = linear

2 = radial

3 = rectangular

4

startX

X coordinate for opacity gradient to begin.

5

startY

Y coordinate for opacity gradient to begin.

6

finishX

X coordinate for opacity gradient to end.

7

finishY

Y coordinate for opacity gradient to end.



```
<html>
<head>
</head>

<body>
  <p>Image Example:</p>

  <img src = "/css/images/logo.png" alt = "CSS Logo"
    style = "Filter: Alpha(Opacity=100,
    FinishOpacity = 0,
    Style = 2,
    StartX = 20,
    StartY = 40,
    FinishX = 0,
    FinishY = 0)" />
  <p>Text Example:</p>

  <div style = "width: 357;
    height: 50;
    font-size: 30pt;
    font-family: Arial Black;
    color: blue;
    Filter: Alpha(Opacity=100, FinishOpacity=0, Style=1, StartX=0, StartY=0, FinishX=588, FinishY=0)" />
</body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Motion Blur

Motion Blur is used to create blurred pictures or text with the direction and strength. The following parameters can be used in this filter –



	<p>True or false. If true, the image is added to the blurred image; and if false, the image is not added to the blurred image.</p>
2	<p>direction</p> <p>The direction of the blur, going clockwise, rounded to 45-degree increments. The default value is 270 (left).</p> <p>0 = Top</p> <p>45 = Top right</p> <p>90 = Right</p> <p>135 = Bottom right</p> <p>180 = Bottom</p> <p>225 = Bottom left</p> <p>270 = Left</p> <p>315 = Top left</p>
3	<p>strength</p> <p>The number of pixels the blur will extend. The default is 5 pixels.</p>

Example

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png" alt = "CSS Logo"
      style = "Filter: Blur(Add = 0, Direction = 225, Strength = 10)">

    <p>Text Example:</p>

    <div style = "width: 357;
      height: 50;
      font-size: 30pt;
      font-family: Arial Black;
```



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

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Chroma Filter

Chroma Filter is used to make any particular color transparent and usually it is used with images. You can use it with scrollbars also. The following parameter can be used in this filter –

Sr.No.	Parameter & Description
1	color The color that you'd like to be transparent.

Example

[Live Demo](#)

```
<html>  
  <head>  
  </head>  
  
  <body>  
    <p>Image Example:</p>  
  
    <img src = "/images/css.gif"  
      alt = "CSS Logo" style = "Filter: Chroma(Color = #FFFFFF)">  
  
    <p>Text Example:</p>  
  
    <div style = "width: 580;
```



```

color: #3300FF;
Filter: Chroma(Color = #3300FF)">CSS Tutorials</div>
</body>
</html>

```

It will produce the following result –

Image Example:



Text Example:

Drop Shadow Effect

Drop Shadow is used to create a shadow of your object at the specified X (horizontal) and Y (vertical) offset and color.

The following parameters can be used in this filter –

Sr.No.	Parameter & Description
1	color The color, in #RRGGBB format, of the dropshadow.
2	offX Number of pixels the drop shadow is offset from the visual object, along the x-axis. Positive integers move the drop shadow to the right, negative integers move the drop shadow to the left.
3	offY Number of pixels the drop shadow is offset from the visual object, along the y-axis. Positive integers move the drop shadow down, negative integers move the drop shadow up.



Example

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png"
      alt = "CSS Logo"
      style = "filter:drop-shadow(2px 2px 1px #FF0000);">

    <p>Text Example:</p>

    <div style = "width: 357;
      height: 50;
      font-size: 30pt;
      font-family: Arial Black;
      color: red;
      filter:drop-shadow(3px 3px 2px #000000);">CSS Tutorials</div>
  </body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Flip Effect



Sr.No.	Parameter & Description
1	FlipH Creates a horizontal mirror image
2	FlipV Creates a vertical mirror image

Example

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png"
      alt = "CSS Logo"
      style = "filter: FlipH">

    <img src = "/css/images/logo.png" alt = "CSS Logo" style = "filter: FlipV">

    <p>Text Example:</p>

    <div style = "width: 300;
      height: 50;
      font-size: 30pt;
      font-family: Arial Black;
      color: red;
      filter: FlipV">CSS Tutorials</div>
  </body>
</html>
```

It will produce the following result –

**tutorialspoint**
SIMPLYEASYLEARNING**tutorialspoint**
SIMPLYEASYLEARNING

Text Example:

CSS Tutorials

Glow Effect

Glow effect is used to create a glow around the object. If it is a transparent image, then glow is created around the opaque pixels of it. The following parameters can be used in this filter –

Sr.No.	Parameter & Description
1	color The color you want the glow to be.
2	strength The intensity of the glow (from 1 to 255).

Example

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png"
      alt = "CSS Logo"
      style = "filter: Chroma(Color = #000000) Glow(Colors=#00FF00, Strength=20)">
```



```
height: 50px;
font-size: 30pt;
font-family: Arial Black;
color: red;
filter: Glow(Color=#00FF00, Strength=20)">CSS Tutorials</div>
</body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Grayscale Effect

Grayscale effect is used to convert the colors of the object to 256 shades of gray. The following parameter is used in this filter –

Sr.No.	Parameter & Description
1	grayscale Converts the colors of the object to 256 shades of gray.

Example

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png"
```

Live Demo



<p>Text Example.</p>

```
<div style = "width: 357;
height: 50;
font-size: 30pt;
font-family: Arial Black;
color: red;
filter: grayscale(50%)">CSS Tutorials</div>
</body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Invert Effect

Invert effect is used to map the colors of the object to their opposite values in the color spectrum, i.e., to create a negative image. The following parameter is used in this filter –

Sr.No.	Parameter & Description
1	Invert Maps the colors of the object to their opposite value in the color spectrum.

Example

```
<html>
<head>
</head>

<body>
```

Live Demo



```
alt = "CSS Logo"
style = "filter: invert(100%)">
```

```
<p>Text Example:</p>
```

```
<div style = "width: 357;
height: 50;
font-size: 30pt;
font-family: Arial Black;
color: red;
filter: invert(100%)">CSS Tutorials</div>
```

```
</body>
```

```
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Mask Effect

Mask effect is used to turn transparent pixels to a specified color and makes opaque pixels transparent. The following parameter is used in this filter –

Sr.No.	Parameter & Description
1	color The color that the transparent areas will become.

Example

```
<html>
<head>
```

Live Demo



<p>Image Example:</p>

```
<img src = "/css/images/logo.png"
  alt = "CSS Logo"
  style = "filter: Chroma(Color = #000000) Mask(Color=#00FF00)">
```

<p>Text Example:</p>

```
<div style = "width: 357;
  height: 50;
  font-size: 30pt;
  font-family: Arial Black;
  color: red;
  filter: Mask(Color=#00FF00)">CSS Tutorials
```

</div>

</body>

</html>

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

Shadow Filter

Shadow filter is used to create an attenuated shadow in the direction and color specified. This is a filter that lies in between Dropshadow and Glow. The following parameters can be used in this filter –

Sr.No.	Parameter & Description
1	color The color that you want the shadow to be.
2	direction



45 = Top right
90 = Right
135 = Bottom right
180 = Bottom
225 = Bottom left
270 = Left
315 = Top left

Example

[Live Demo](#)

```
<html>
  <head>
  </head>

  <body>
    <p>Image Example:</p>

    <img src = "/css/images/logo.png"
      alt = "CSS Logo"
      style = "filter: Chroma(Color = #000000) Shadow(Color=#00FF00, Direction=225)">

    <p>Text Example:</p>

    <div style = "width: 357;
      height: 50;
      font-size: 30pt;
      font-family:
        Arial Black;
      color: red;
      filter: Shadow(Color=#0000FF, Direction=225)">CSS Tutorials
    </div>
  </body>
</html>
```

It will produce the following result –



Text Example:

CSS Tutorials

Wave Effect

Wave effect is used to give the object a sine wave distortion to make it look wavy. The following parameters can be used in this filter –

Sr.No.	Parameter & Description
1	add A value of 1 adds the original image to the waved image, 0 does not.
2	freq The number of waves.
3	light The strength of the light on the wave (from 0 to 100).
4	phase At what degree the sine wave should start (from 0 to 100).
5	strength The intensity of the wave effect.

Example

```
<html>
<head>
</head>
```

[Live Demo](#)



```
<img src = "/css/images/logo.png"
  alt = "CSS Logo"
  style = "filter: Chroma(Color = #000000)
  Wave(Add=0, Freq=1, LightStrength=10, Phase=220, Strength=10)">

<p>Text Example:</p>

<div style = "width: 357;
  height: 50;
  font-size: 30pt;
  font-family: Arial Black;
  color: red;
  filter: Wave(Add=0, Freq=1, LightStrength=10, Phase=20, Strength=20)">CSS Tutorial
</div>
</body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

X-Ray Effect

X-Ray effect grayscales and flattens the color depth. The following parameter is used in this filter:

Sr.No.	Parameter & Description
1	xray Grayscales and flattens the color depth.

Example



```
</head>

<body>
  <p>Image Example:</p>

  <img src = "/css/images/logo.png"
    alt = "CSS Logo"
    style = "filter: Xray">

  <p>Text Example:</p>

  <div style = "width: 357;
    height: 50;
    font-size: 30pt;
    font-family: Arial Black;
    color: red;
    filter: Xray">CSS Tutorials
  </div>
</body>
</html>
```

It will produce the following result –

Image Example:



Text Example:

CSS Tutorials

CSS - Media Types

Given below is an example –

```
<style tyle = "text/css">
  <!--
    @media print {
      body { font-size: 10pt }
    }
  </style>
```



```

}
@media screen, print {
    body { line-height: 1.2 }
}
-->
</style>

```

The Document Language

In HTML 4.0, the *media* attribute on the LINK element specifies the target media of an external style sheet –

Following is an example –

```

<style tyle = "text/css">
  <!--
    <!doctype html public "-//w3c//dtd html 4.0//en">
    <html>
      <head>
        <title>link to a target medium</title>
        <link rel = "stylesheet" type = "text/css" media = "print,
          handheld" href = "foo.css">
      </head>

      <body>
        <p>the body...
      </body>
    </html>
  -->
</style>

```

Recognized Media Types

The names chosen for CSS media types reflect target devices for which the relevant properties make sense. They give a sense of what device the media type is meant to refer to. Given below is a list of various media types –

Sr.No.	Value & Description
1	all Suitable for all devices.
2	aural Intended for speech synthesizers.



	Intended for braille tactile feedback devices.
4	embossed Intended for paged braille printers.
5	handheld Intended for handheld devices (typically small screen, monochrome, limited bandwidth).
6	print Intended for paged, opaque material and for documents viewed on screen in print preview mode. Please consult the section on paged media.
7	projection Intended for projected presentations, for example projectors or print to transparencies. Please consult the section on paged media.
8	screen Intended primarily for color computer screens.
9	tty Intended for media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities.
10	tv Intended for television-type devices.

NOTE – Media type names are case-insensitive.

CSS Paged Media - @page Rule

The CSS2 defines a "page box", a box of finite dimensions in which content is rendered. The page box is a rectangular region that contains two areas –



You can specify the dimensions, orientation, margins, etc., of a page box within an `@page` rule. The dimensions of the page box are set with the 'size' property. The dimensions of the page area are the dimensions of the page box minus the margin area.

For example, the following `@page` rule sets the page box size to 8.5 × 11 inches and creates '2cm' margin on all sides between the page box edge and the page area –

```
<style type = "text/css">
  <!--
    @page { size:8.5in 11in; margin: 2cm }
  -->
</style>
```

You can use the *margin*, *margin-top*, *margin-bottom*, *margin-left*, and *margin-right* properties within the `@page` rule to set margins for your page.

Finally, the *marks* property is used within the `@page` rule to create crop and registration marks outside the page box on the target sheet. By default, no marks are printed. You may use one or both of the *crop* and *cross* keywords to create crop marks and registration marks, respectively, on the target print page.

Setting Page Size

The *size* property specifies the size and orientation of a page box. There are four values which can be used for page size –

auto – The page box will be set to the size and orientation of the target sheet.

landscape – Overrides the target's orientation. The page box is the same size as the target, and the longer sides are horizontal.

portrait – Overrides the target's orientation. The page box is the same size as the target, and the shorter sides are horizontal.

length – Length values for the 'size' property create an absolute page box. If only one length value is specified, it sets both the width and height of the page box. Percentage values are not allowed for the 'size' property.

In the following example, the outer edges of the page box will align with the target. The percentage value on the 'margin' property is relative to the target size so if the target sheet dimensions are 21.0cm × 29.7cm (i.e., A4), the margins are 2.10cm and 2.97cm.

```
<style type = "text/css">
  <!--
    @page {
      size: auto; /* auto is the initial value */
      margin: 10%;
    }
  -->
</style>
```



The following example sets the width of the page box to be 8.5 inches and the height to be 11 inches. The page box in this example requires a target sheet size of 8.5" × 11" or larger.

```
<style type = "text/css">
  <!--
    @page {
      size: 8.5in 11in; /* width height */
    }
  -->
</style>
```

Once you create a named page layout, you can use it in your document by adding the page property to a style that is later applied to an element in your document. For example, this style renders all the tables in your document on landscape pages –

```
<style type = "text/css">
  <!--
    @page { size : portrait }
    @page rotated { size : landscape }
    table { page : rotated }
  -->
</style>
```

Due to the above rule, while printing, if the browser encounters a <table> element in your document and the current page layout is the default portrait layout, it starts a new page and prints the table on a landscape page.

Left, Right, and First pages

When printing double-sided documents, the page boxes on left and right pages should be different. It can be expressed through two CSS pseudo-classes as follows –

```
<style type = "text/css">
  <!--
    @page :left {
      margin-left: 4cm;
      margin-right: 3cm;
    }

    @page :right {
      margin-left: 3cm;
      margin-right: 4cm;
    }
  -->
</style>
```



```
@page { margin: 2cm } /* ALL margins set to 2cm */

@page :first {
    margin-top: 10cm    /* Top margin on first page 10cm */
}
-->
</style>
```

Controlling Pagination

Unless you specify otherwise, page breaks occur only when the page format changes or when the content overflows the current page box. To otherwise force or suppress page breaks, use the *page-break-before*, *page-break-after*, and *page-break-inside* properties.

Both the *page-break-before* and *page-break-after* accept the *auto*, *always*, *avoid*, *left*, and *right* keywords.

The keyword *auto* is the default, it lets the browser generate page breaks as needed. The keyword *always* forces a page break before or after the element, while *avoid* suppresses a page break immediately before or after the element. The *left* and *right* keywords force one or two page breaks, so that the element is rendered on a left-hand or right-hand page.

Using pagination properties is quite straightforward. Suppose your document has level-1 headers start new chapters with level-2 headers to denote sections. You'd like each chapter to start on a new, right-hand page, but you don't want section headers to be split across a page break from the subsequent content. You can achieve this using following rule –

```
<style type = "text/css">
  <!--
    h1 { page-break-before : right }
    h2 { page-break-after : avoid }
  -->
</style>
```

Use only the *auto* and *avoid* values with the *page-break-inside* property. If you prefer that your tables not be broken across pages if possible, you would write the rule –

```
<style type = "text/css">
  <!--
    table { page-break-inside : avoid }
  -->
</style>
```

Controlling Widows and Orphans



Most printers try to leave at least two or more lines of text at the top or bottom of each page.

The **orphans** property specifies the minimum number of lines of a paragraph that must be left at the bottom of a page.

The **widows** property specifies the minimum number of lines of a paragraph that must be left at the top of a page.

Here is the example to create 4 lines at the bottom and 3 lines at the top of each page –

```
<style type = "text/css">
  <!--
    @page{orphans:4; widows:2;}
  -->
</style>
```

CSS - Aural Media

- Medical documentation

When using aural properties, the canvas consists of a three-dimensional physical space (sound surrounds) and a temporal space (one may specify sounds before, during, and after other sounds).

The CSS properties also allow you to vary the quality of synthesized speech (voice type, frequency, inflection, etc.).

Here is an example –

```
<html>
  <head>
    <style type = "text/css">
      h1, h2, h3, h4, h5, h6 {
        voice-family: paul;
        stress: 20;
        richness: 90;
        cue-before: url("../audio/pop.au");
      }
      p {
        azimuth:center-right;
      }
    </style>
  </head>

  <body>

    <h1>Tutorialspoint.com</h1>
    <h2>Tutorialspoint.com</h2>
    <h3>Tutorialspoint.com</h3>
```



```
<p>tutorialspoint.com</p>
```

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

It will produce the following result –

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

It will direct the speech synthesizer to speak headers in a voice (a kind of audio font) called "paul", on a flat tone, but in a very rich voice. Before speaking the headers, a sound sample will be played from the given URL.

Paragraphs with class 'heidi' will appear to come from front left (if the sound system is capable of spatial audio), and paragraphs of class 'peter' from the right.

Now we will see the various properties related to aural media.

The **azimuth** property sets, where the sound should come from horizontally.

The **elevation** property sets, where the sound should come from vertically.

The **cue-after** specifies a sound to be played after speaking an element's content to delimit it from other.

The **cue-before** specifies a sound to be played before speaking an element's content to delimit it from other.

The **cue** is a shorthand for setting cue-before and cue-after.

The **pause-after** specifies a pause to be observed after speaking an element's content.

The **pause-before** specifies a pause to be observed before speaking an element's content.

The **pause** is a shorthand for setting pause-before and pause-after.



The **play-during** specifies a sound to be played as a background while an element's content is spoken.

The **richness** specifies the richness, or brightness, of the speaking voice.

The **speak** specifies whether text will be rendered aurally and if so, in what manner.

The **speak-numeral** controls how numerals are spoken.

The **speak-punctuation** specifies how punctuation is spoken.

The **speech-rate** specifies the speaking rate.

The **stress** specifies the height of "local peaks" in the intonation contour of a voice.

The **voice-family** specifies the prioritized list of voice family names.

The **volume** refers to the median volume of the voice.

The azimuth Property

The azimuth property sets where the sound should come from horizontally. The possible values are listed below –

angle – Position is described in terms of an angle within the range *-360deg* to *360deg*. The value *0deg* means directly ahead in the center of the sound stage. *90deg* is to the right, *180deg* behind, and *270deg* (or, equivalently and more conveniently, *-90deg*) to the left.

left-side – Same as '270deg'. With 'behind', '270deg'.

far-left – Same as '300deg'. With 'behind', '240deg'.

left – Same as '320deg'. With 'behind', '220deg'.

center-left – Same as '340deg'. With 'behind', '200deg'.

center – Same as '0deg'. With 'behind', '180deg'.

center-right – Same as '20deg'. With 'behind', '160deg'.

right – Same as '40deg'. With 'behind', '140deg'.

far-right – Same as '60deg'. With 'behind', '120deg'.

right-side – Same as '90deg'. With 'behind', '90deg'.

leftwards – Moves the sound to the left and relative to the current angle. More precisely, subtracts 20 degrees.

rightwards – Moves the sound to the right, relative to the current angle. More precisely, adds 20 degrees.

Here is an example –

```
<style type = "text/css">
  <!--
```



```
-->
</style>
```

The elevation Property

The elevation property sets where the sound should come from vertically. The possible values are as follows –

angle – Specifies the elevation as an angle, between *-90deg* and *90deg*. *0deg* means on the forward horizon, which loosely means level with the listener. *90deg* means directly overhead and *-90deg* means directly below.

below – Same as *'-90deg'*.

level – Same as *'0deg'*.

above – Same as *'90deg'*.

higher – Adds 10 degrees to the current elevation.

lower – Subtracts 10 degrees from the current elevation.

Here is an example –

```
<style type = "text/css">
  <!--
    h1 { elevation: above }
    tr.a { elevation: 60deg }
    tr.b { elevation: 30deg }
    tr.c { elevation: level }
  -->
</style>
```

The cue-after Property

The cue-after property specifies a sound to be played after speaking an element's content to delimit it from other. The possible values include –

url – The URL of a sound file to be played.

none – Nothing has to be played.

Here is an example –

```
<style type = "text/css">
  <!--
    a { cue-after: url("dong.wav");}
    h1 { cue-after: url("pop.au"); }
```



The cue-before Property

This property specifies a sound to be played before speaking an element's content to delimit it from other. The possible values are –

url – The URL of a sound file to be played.

none – Nothing has to be played.

Here is an example –

```
<style type = "text/css">
  <!--
    a {cue-before: url("bell.aiff");}
    h1 {cue-before: url("pop.au"); }
  -->
</style>
```

The cue Property

The cue property is a shorthand for setting *cue-before* and *cue-after*. If two values are given, the first value is *cue-before* and the second is *cue-after*. If only one value is given, it applies to both properties.

For example, the following two rules are equivalent –

```
<style type = "text/css">
  <!--
    h1 {cue-before: url("pop.au"); cue-after: url("pop.au") }
    h1 {cue: url("pop.au") }
  -->
</style>
```

The pause-after Property

This property specifies a pause to be observed after speaking an element's content. The possible values are –

time – Expresses the pause in absolute time units (seconds and milliseconds).

percentage – Refers to the inverse of the value of the *speech-rate* property. For example, if the speech-rate is 120 words per minute (i.e. a word takes half a second, or 500ms), then a *pause-after* of 100% means a pause of 500 ms and a *pause-after* of 20% means 100ms.

The pause-before Property



percentage – Refers to the inverse of the value of the *speech-rate* property. For example, if the speech-rate is 120 words per minute (i.e. a word takes half a second, or 500ms), then a *pause-before* of 100% means a pause of 500 ms and a *pause-before* of 20% means 100ms.

The pause Property

This property is a shorthand for setting *pause-before* and *pause-after*. If two values are given, the first value is *pause-before* and the second is *pause-after*.

Here is an example –

```
<style type = "text/css">
  <!--
    /* pause-before: 20ms; pause-after: 20ms */
    h1 { pause : 20ms }

    /* pause-before: 30ms; pause-after: 40ms */
    h2 { pause : 30ms 40ms }

    /* pause-before: ?; pause-after: 10ms */
    h3 { pause-after : 10ms }
  -->
</style>
```

The pitch Property

This property specifies the average pitch (a frequency) of the speaking voice. The average pitch of a voice depends on the voice family. For example, the average pitch for a standard male voice is around 120Hz, but for a female voice, it's around 210Hz. The possible values are –

frequency – Specifies the average pitch of the speaking voice in hertz (Hz).

x-low, low, medium, high, x-high – These values do not map to absolute frequencies since these values depend on the voice family.

The pitch-range Property

This property specifies variation in average pitch. The possible values are –

number – A value between '0' and '100'. A pitch range of '0' produces a flat, monotonic voice. A pitch range of 50 produces normal inflection. Pitch ranges greater than 50 produce animated voices.

The play-during Property



content is spoken.

mix – When present, this keyword means that the sound inherited from the parent element's *play-during* property continues to play and the sound designated by the *uri* is mixed with it. If *mix* is not specified, the element's background sound replaces the parent's.

repeat – When present, this keyword means that the sound will repeat if it is too short to fill the entire duration of the element. Otherwise, the sound plays once and then stops.

auto – The sound of the parent element continues to play.

none – This keyword means that there is silence.

Here is an example –

```
<style type = "text/css">
  <!--
    blockquote.sad { play-during: url("violins.aiff") }
    blockquote.q   { play-during: url("harp.wav") mix }
    span.quiet     { play-during: none }
  -->
</style>
```

The richness Property

This property specifies the richness or brightness of the speaking voice. The possible values are –

number – A value between '0' and '100'. The higher the value, the more the voice will carry. A lower value will produce a soft, mellifluous voice.

The speak Property

This property specifies whether text will be rendered aurally and if so, in what manner. The possible values are –

none – Suppresses aural rendering so that the element requires no time to render.

normal – Uses language-dependent pronunciation rules for rendering an element and its children.

spell-out – Spells the text one letter at a time.

Note the difference between an element whose 'volume' property has a value of 'silent' and an element whose 'speak' property has the value 'none'. The former takes up the same time as if it had been spoken, including any pause before and after the element, but no sound is generated. The latter requires no time and is not rendered.

The speak-numeral Property



continuous – Speak the numeral as a full number. Thus, "237" is spoken "Two hundred thirty seven". Word representations are language-dependent.

The speak-punctuation Property

This property specifies how punctuation is spoken. The possible values are –

code – Punctuation such as semicolons, braces, and so on are to be spoken literally.

none – Punctuation is not to be spoken, but instead rendered naturally as various pauses.

The speech-rate property

This property specifies the speaking rate. Note that both absolute and relative keyword values are allowed. The possible values are –

number – Specifies the speaking rate in words per minute.

x-slow – Same as 80 words per minute.

slow – Same as 120 words per minute.

medium – Same as 180 - 200 words per minute.

fast – Same as 300 words per minute.

x-fast – Same as 500 words per minute.

faster – Adds 40 words per minute to the current speech rate.

slower – Subtracts 40 words per minutes from the current speech rate.

The stress Property

This property specifies the height of "local peaks" in the intonation contour of a voice. English is a stressed language, and different parts of a sentence are assigned primary, secondary, or tertiary stress. The possible values are –

number – A value between '0' and '100'. The meaning of values depends on the language being spoken. For example, a level of '50' for a standard, English-speaking male voice (average pitch = 122Hz), speaking with normal intonation and emphasis would have a different meaning than '50' for an Italian voice.

The voice-family Property

The value is a comma-separated, prioritized list of voice family names. It can have following values –

generic-voice – Values are voice families. Possible values are 'male', 'female', and 'child'.

specific-voice – Values are specific instances (e.g., comedian, trinoids, carlos, lani).



```
h1 { voice-family: announcer, male }
p.part.romeo { voice-family: romeo, male }
p.part.juliet { voice-family: juliet, female }
-->
</style>
```

The volume Property

Volume refers to the median volume of the voice. It can have following values –

numbers – Any number between '0' and '100'. '0' represents the minimum audible volume level and 100 corresponds to the maximum comfortable level.

percentage – These values are calculated relative to the inherited value, and are then clipped to the range '0' to '100'.

silent – No sound at all. The value '0' does not mean the same as 'silent'.

x-soft – Same as '0'.

soft – Same as '25'.

medium – Same as '50'.

loud – Same as '75'.

x-loud – Same as '100'.

Here is an example –

```
<style type = "text/css">
  <!--
    P.goat { volume: x-soft }
  -->
</style>
```

Paragraphs with class **goat** will be very soft.

CSS Printing - @media Rule

```
} @media print { p.bodyText {font-family:georgia, times, serif;} } @media screen, print { p.bodyText {font-size:10pt} } --> </style>
```

If you are defining your style sheet in a separate file, then you can also use the media attribute when linking to an external style sheet –

```
<link rel = "stylesheet" type = "text/css" media = "print" href = "mystyle.css">
```



CSS is pivotal to the future of Web documents and will be supported by most browsers.

CSS is more exact than tables, allowing your document to be viewed as you intended, regardless of the browser window.

Keeping track of nested tables can be a real pain. CSS rules tend to be well organized, easily read, and easily changed.

Finally, we would suggest you to use whichever technology makes sense to you and use what you know or what presents your documents in the best way.

CSS also provides *table-layout* property to make your tables load much faster. Following is an example –

```
<table style = "table-layout:fixed;width:600px;">
  <tr height = "30">
    <td width = "150">CSS table layout cell 1</td>
    <td width = "200">CSS table layout cell 2</td>
    <td width = "250">CSS table layout cell 3</td>
  </tr>
</table>
```

You will notice the benefits more on large tables. With traditional HTML, the browser had to calculate every cell before finally rendering the table. When you set the table-layout algorithm to *fixed*, however, it only needs to look at the first row before rendering the whole table. It means your table will need to have fixed column widths and row heights.

Sample Column Layout

Here are the steps to create a simple Column Layout using CSS –

Set the margin and padding of the complete document as follows –

```
<style style = "text/css">
  <!--
    body {
      margin:9px 9px 0 9px;
      padding:0;
      background:#FFF;
    }
  -->
</style>
```

Now, we will define a column with yellow color and later, we will attach this rule to a <div> –

```
<style style = "text/css">
  <!--
```



```
</style>
```

Upto this point, we will have a document with yellow body, so let us now define another division inside level0 –

```
<style style = "text/css">
  <!--
    #level1 {
      margin-left:143px;
      padding-left:9px;
      background:#FFF;
    }
  -->
</style>
```

Now, we will nest one more division inside level1, and we will change just background color –

```
<style style = "text/css">
  <!--
    #Level2 {
      background:#FFF3AC;
    }
  -->
</style>
```

Finally, we will use the same technique, nest a level3 division inside level2 to get the visual layout for the right column –

```
<style style = "text/css">
  <!--
    #Level3 {
      margin-right:143px;
      padding-right:9px;
      background:#FFF;
    }
    #main {
      background:#CCC;
    }
  -->
</style>
```

Complete the source code as follows –

```
<style style = "text/css">
  body {
```

[Live Demo](#)



```
}  
  
#Level0 {background:#FC0;}  
  
#Level1 {  
    margin-left:143px;  
    padding-left:9px;  
    background:#FFF;  
}  
  
#Level2 {background:#FFF3AC;}  
  
#Level3 {  
    margin-right:143px;  
    padding-right:9px;  
    background:#FFF;  
}  
  
#main {background:#CCC;}  
</style>  
<body>  
    <div id = "level0">  
        <div id = "level1">  
            <div id = "level2">  
                <div id = "level3">  
                    <div id = "main">  
                        Final Content goes here...  
                    </div>  
                </div>  
            </div>  
        </div>  
    </div>  
</body>
```

Similarly, you can add a top navigation bar or an ad bar at the top of the page.

It will produce the following result –

Final Content goes here...

CSS - Validations

W3C CSS Validator (World Wide Web Consortium), This validator checks your css by either file upload, direct input, or using URI - one page at a time. This validator helps you to locate all the errors in your CSS.



The WDG CSS check validator, lets you validate your CSS by direct input, file upload, and using URI. Errors will be listed by line and column numbers if you have any. Errors usually come with links to explain the reason of error.

A CSS validator checks your Cascading Style Sheets to make sure that they comply with the CSS standards set by the W3 Consortium. There are a few validators which will also tell you which CSS features are supported by which browsers (since not all browsers are equal in their CSS implementation).

Why Validate Your HTML Code?

There are a number of reasons why you should validate your code. But major ones are –

- It Helps Cross-Browser, Cross-Platform, and Future Compatibility.

- A good quality website increases search engine visibility.

- Professionalism: As a web developer, your code should not raise errors while seen by the visitors.

CSS3 - Tutorial

CSS3 is collaboration of CSS2 specifications and new specifications, we can call this collaboration is **module**. Some of the modules are shown below –

- Selectors
- Box Model
- Backgrounds
- Image Values and Replaced Content
- Text Effects
- 2D Transformations
- 3D Transformations
- Animations
- Multiple Column Layout
- User Interface

CSS3 - Rounded Corners

```
}
```

The following table shows the possible values for Rounded corners as follows –

Sr.No.	Value & Description
--------	---------------------



2	border-top-left-radius Use this element for setting the boarder of top left corner
3	border-top-right-radius Use this element for setting the boarder of top right corner
4	border-bottom-right-radius Use this element for setting the boarder of bottom right corner
5	border-bottom-left-radius Use this element for setting the boarder of bottom left corner

Example

This property can have three values. The following example uses both the values –

[Live Demo](#)

```
<html>
  <head>
    <style>
      #rcorners1 {
        border-radius: 25px;
        background: #8AC007;
        padding: 20px;
        width: 200px;
        height: 150px;
      }
      #rcorners2 {
        border-radius: 25px;
        border: 2px solid #8AC007;
        padding: 20px;
        width: 200px;
        height: 150px;
      }
      #rcorners3 {
        border-radius: 25px;
        background: url(/css/images/logo.png);
        background-position: left top;
        background-repeat: repeat;
      }
    </style>
  </head>
</html>
```

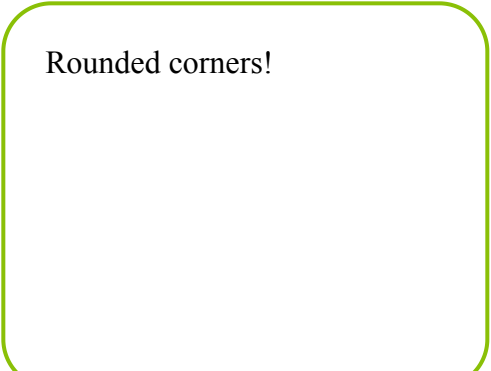


```
</style>
</head>

<body>
  <p id = "rcorners1">Rounded corners!</p>
  <p id = "rcorners2">Rounded corners!</p>
  <p id = "rcorners3">Rounded corners!</p>
</body>
</html>
```

It will produce the following result –

Rounded corners!



Rounded corners!

Each Corner property

We can specify the each corner property as shown below example –

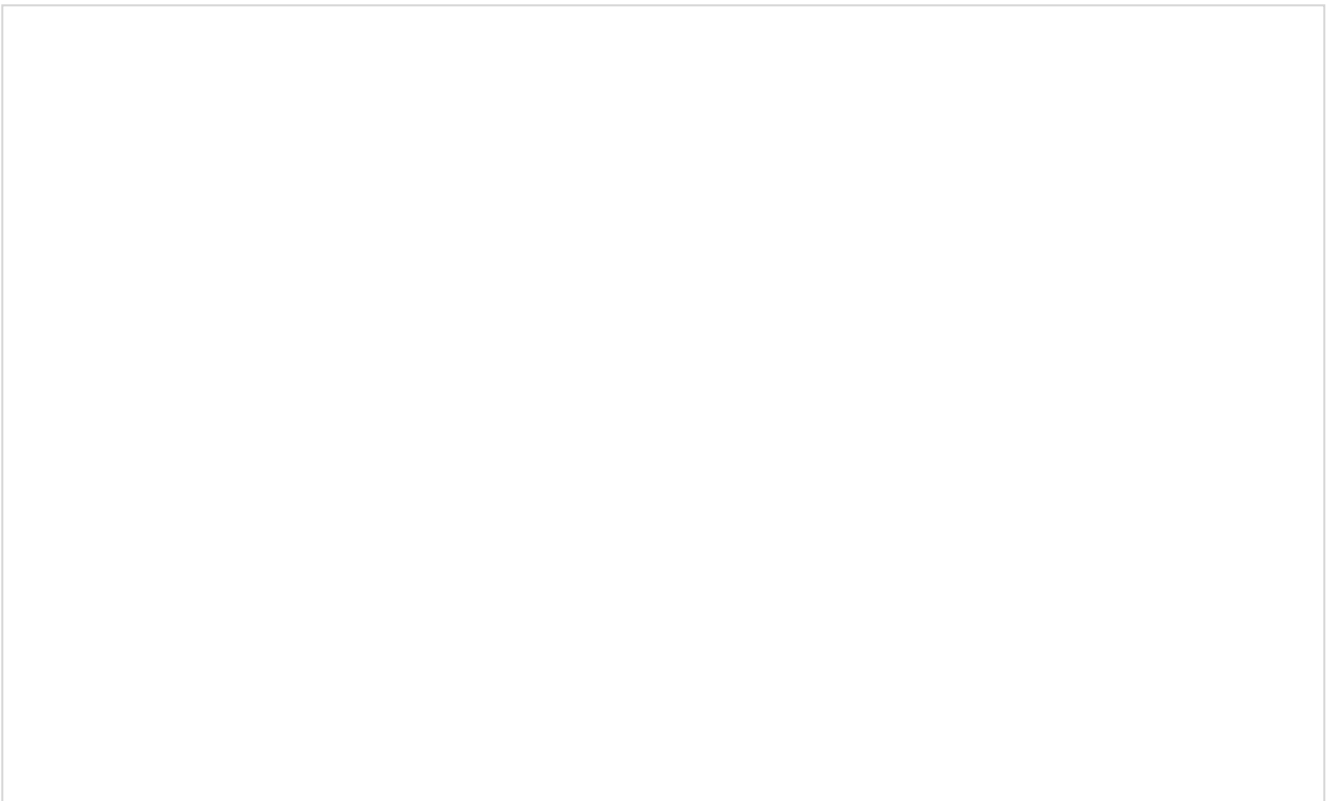
```
<html>
  <head>
    <style>
      #rcorners1 {
        border-radius: 15px 50px 30px 5px;
        background: #a44170;
        padding: 20px;
        width: 100px;
```

Live Demo



```
border-radius: 15px 50px 50px;  
background: #a44170;  
padding: 20px;  
width: 100px;  
height: 100px;  
}  
#rcorners3 {  
border-radius: 15px 50px;  
background: #a44170;  
padding: 20px;  
width: 100px;  
height: 100px;  
}  
</style>  
</head>  
  
<body>  
  <p id = "rcorners1"></p>  
  <p id = "rcorners2"></p>  
  <p id = "rcorners3"></p>  
</body>  
</body>
```

It will produce the following result –





Sr.No.	Value & Description
1	border-image-source Used to set the image path
2	border-image-slice Used to slice the boarder image
3	border-image-width Used to set the boarder image width
4	border-image-repeat Used to set the boarder image as rounded, repeated and stretched

Example

Following is the example which demonstrates to set image as a border for elements.

[Live Demo](#)

```
<html>
  <head>
    <style>
      #borderimg1 {
        border: 10px solid transparent;
        padding: 15px;
        border-image-source: url(/css/images/border.png);
        border-image-repeat: round;
        border-image-slice: 30;
        border-image-width: 10px;
      }
      #borderimg2 {
        border: 10px solid transparent;
        padding: 15px;
        border-image-source: url(/css/images/border.png);
        border-image-repeat: round;
        border-image-slice: 30;
        border-image-width: 20px;
      }
      #borderimg3 {
        border: 10px solid transparent;
```



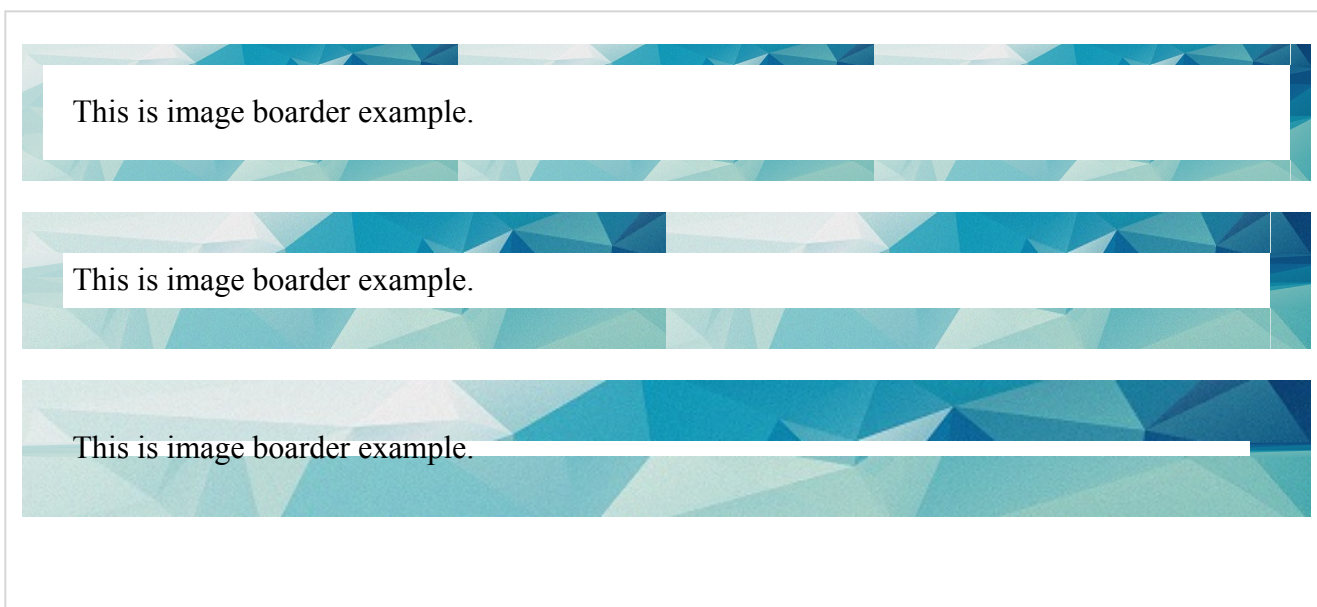
```

border-image-size: 30,
border-image-width: 30px;
}
</style>
</head>

<body>
  <p id = "borderimg1">This is image boarder example.</p>
  <p id = "borderimg2">This is image boarder example.</p>
  <p id = "borderimg3">This is image boarder example.</p>
</body>
</html>

```

It will produce the following result –



CSS3 - Multi Background

```

}

```

The most commonly used values are shown below –

Sr.No.	Value & Description
1	background Used to setting all the background image properties in one section
2	background-clip Used to declare the painting area of the background



	Used to specify the background image
4	background-origin Used to specify position of the background images
5	background-size Used to specify size of the background images

Example

Following is the example which demonstrate the multi background images.

[Live Demo](#)

```

<html>
  <head>
    <style>
      #multibackground {
        background-image: url(/css/images/logo.png), url(/css/images/border.png);
        background-position: left top, left top;
        background-repeat: no-repeat, repeat;
        padding: 75px;
      }
    </style>
  </head>

  <body>

    <div id = "multibackground">
      <h1>www.tutorialspoint.com</h1>
      <p>
        Tutorials Point originated from the idea that there exists a class of
        readers who respond better to online content and prefer to learn new
        skills at their own pace from the comforts of their drawing rooms.
        The journey commenced with a single tutorial on HTML in 2006 and elated
        by the response it generated, we worked our way to adding fresh tutorials
        to our repository which now proudly flaunts a wealth of tutorials and
        allied articles on topics ranging from programming languages to web designing
        to academics and much more..
      </p>
    </div>

  </body>
</html>

```



www.tutorialspoint.com

Tutorials Point originated from the idea that there exists a class of readers who respond better to online content and prefer to learn new skills at their own pace from the comforts of their drawing rooms. The journey commenced with a single tutorial on HTML in 2006 and elated by the response it generated, we worked our way to adding fresh tutorials to our repository which now proudly flaunts a wealth of tutorials and allied articles on topics ranging from programming languages to web designing to academics and much more..

Size of Multi background

Multi background property is accepted to add different sizes for different images. A sample syntax is as shown below –

```
#multibackground {  
    background: url(/css/imalges/logo.png) left top no-repeat, url(/css/images/boarder.png) r  
    background-size: 50px, 130px, auto;  
}
```

As shown above an example, each image is having specific sizes as 50px, 130px and auto size.

CSS3 - Colors

```
#d1 {background-color: rgba(255, 0, 0, 0.5);}  
#d2 {background-color: rgba(0, 255, 0, 0.5);}  
#d3 {background-color: rgba(0, 0, 255, 0.5);}
```

HSL stands for **hue, saturation, lightness**. Here Hue is a degree on the color wheel, saturation and lightness are percentage values between 0 to 100%. A Sample syntax of HSL as shown below –



HSLA stands for **hue, saturation, lightness and alpha**. Alpha value specifies the opacity as shown RGBA. A Sample syntax of HSLA as shown below –

```
#g1 {background-color: hsla(120, 100%, 50%, 0.3);}
#g2 {background-color: hsla(120, 100%, 75%, 0.3);}
#g3 {background-color: hsla(120, 100%, 25%, 0.3);}
```

opacity is a thinner paints need black added to increase opacity. A sample syntax of opacity is as shown below –

```
#g1 {background-color:rgb(255,0,0);opacity:0.6;}
#g2 {background-color:rgb(0,255,0);opacity:0.6;}
#g3 {background-color:rgb(0,0,255);opacity:0.6;}
```

The following example shows rgba color property.

[Live Demo](#)

```
<html>
  <head>
    <style>
      #p1 {background-color:rgba(255,0,0,0.3);}
      #p2 {background-color:rgba(0,255,0,0.3);}
      #p3 {background-color:rgba(0,0,255,0.3);}
    </style>
  </head>

  <body>
    <p>RGBA colors:</p>
    <p id = "p1">Red</p>
    <p id = "p2">Green</p>
    <p id = "p3">Blue</p>
  </body>
</html>
```

It will produce the following result –

RGBA colors:

Red

Green

Blue



```
<html>
<head>
  <style>
    #g1 {background-color:hsl(120, 100%, 50%);}
    #g2 {background-color:hsl(120,100%,75%);}
    #g3 {background-color:hsl(120,100%,25%);}
  </style>
</head>

<body>
  <p>HSL colors:</p>
  <p id = "g1">Green</p>
  <p id = "g2">Normal Green</p>
  <p id = "g3">Dark Green</p>
</body>
</html>
```

It will produce the following result –

HSL colors:

Green

Normal Green

Dark Green

The following example shows HSLA color property.

Live Demo

```
<html>
<head>
  <style>
    #d1 {background-color:hsla(120,100%,50%,0.3);}
    #d2 {background-color:hsla(120,100%,75%,0.3);}
    #d3 {background-color:hsla(120,100%,25%,0.3);}
  </style>
</head>

<body>
  <p>HSLA colors:</p>
  <p id = "d1">Less opacity green</p>
  <p id = "d2">Green</p>
  <p id = "d3">Green</p>
</body>
</html>
```



Less opacity green

Green

Green

The following example shows Opacity property.

[Live Demo](#)

```
<html>
  <head>
    <style>
      #m1 {background-color:rgb(255,0,0);opacity:0.6;}
      #m2 {background-color:rgb(0,255,0);opacity:0.6;}
      #m3 {background-color:rgb(0,0,255);opacity:0.6;}
    </style>
  </head>

  <body>
    <p>HSLA colors:</p>
    <p id = "m1">Red</p>
    <p id = "m2">Green</p>
    <p id = "m3">Blue</p>
  </body>
</html>
```

It will produce the following result –

HSLA colors:

Red

Green

Blue

CSS3 - Gradients

Linear gradients

Linear gradients are used to arrange two or more colors in linear formats like top to bottom.

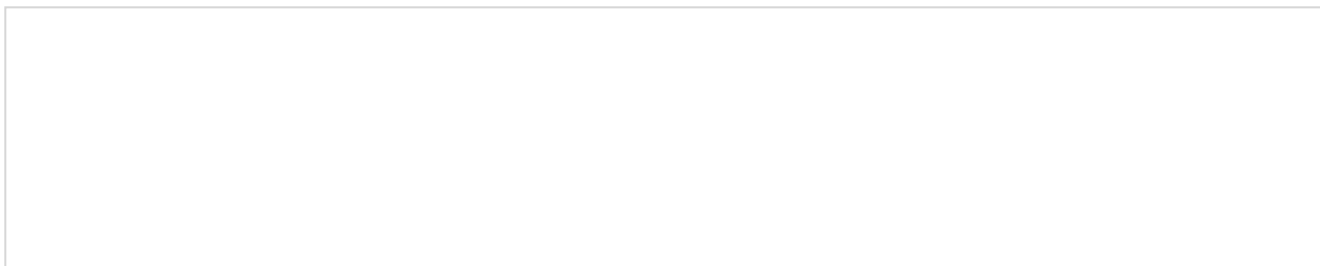
Top to bottom



```
<style>
  #grad1 {
    height: 100px;
    background: -webkit-linear-gradient(pink,green);
    background: -o-linear-gradient(pink,green);
    background: -moz-linear-gradient(pink,green);
    background: linear-gradient(pink,green);
  }
</style>
</head>

<body>
  <div id = "grad1"></div>
</body>
</html>
```

It will produce the following result –



Left to right

Live Demo

```
<html>
  <head>
    <style>
      #grad1 {
        height: 100px;
        background: -webkit-linear-gradient(left, red , blue);
        background: -o-linear-gradient(right, red, blue);
        background: -moz-linear-gradient(right, red, blue);
        background: linear-gradient(to right, red , blue);
      }
    </style>
  </head>

  <body>
    <div id = "grad1"></div>
  </body>
</html>
```



Diagonal

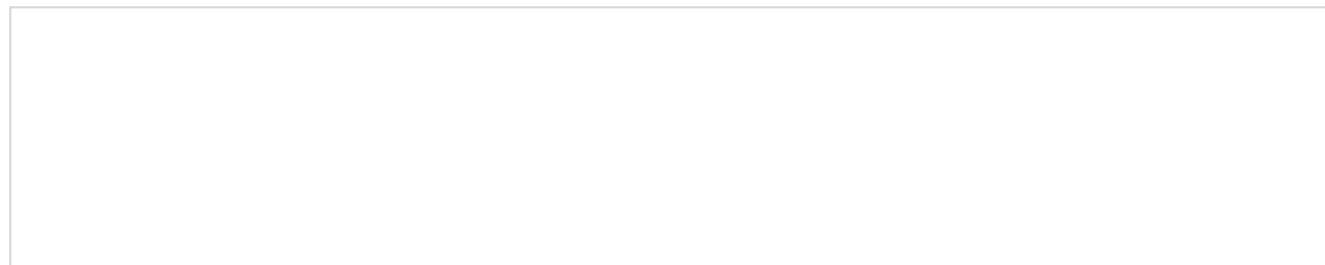
Diagonal starts at top left and right button.

[Live Demo](#)

```
<html>
  <head>
    <style>
      #grad1 {
        height: 100px;
        background: -webkit-linear-gradient(left top, red , blue);
        background: -o-linear-gradient(bottom right, red, blue);
        background: -moz-linear-gradient(bottom right, red, blue);
        background: linear-gradient(to bottom right, red , blue);
      }
    </style>
  </head>

  <body>
    <div id = "grad1"></div>
  </body>
</html>
```

It will produce the following result –



Multi color

[Live Demo](#)

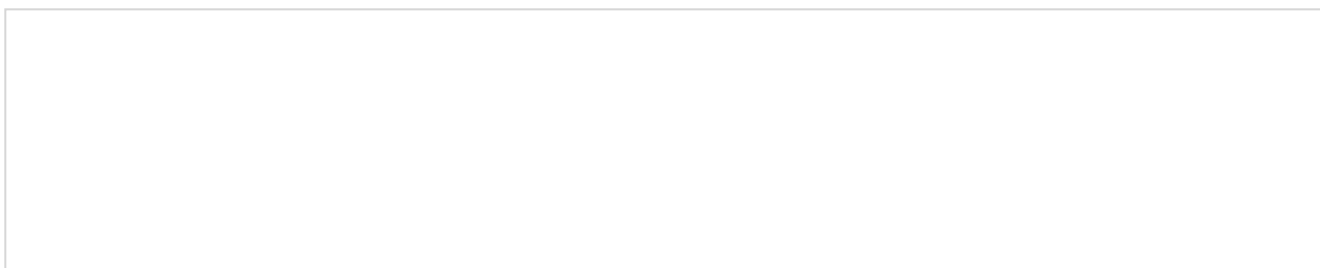
```
<html>
  <head>
    <style>
      #grad2 {
        height: 100px;
```



```
background: linear-gradient(red, orange, yellow, red, blue, green, pink);
    }
</style>
</head>

<body>
    <div id = "grad2"></div>
</body>
</html>
```

It will produce the following result –



CSS3 Radial Gradients

Radial gradients appears at center.

Live Demo

```
<html>
  <head>
    <style>
      #grad1 {
        height: 100px;
        width: 550px;
        background: -webkit-radial-gradient(red 5%, green 15%, pink 60%);
        background: -o-radial-gradient(red 5%, green 15%, pink 60%);
        background: -moz-radial-gradient(red 5%, green 15%, pink 60%);
        background: radial-gradient(red 5%, green 15%, pink 60%);
      }
    </style>
  </head>

  <body>
    <div id = "grad1"></div>
  </body>
</html>
```

It will produce the following result –



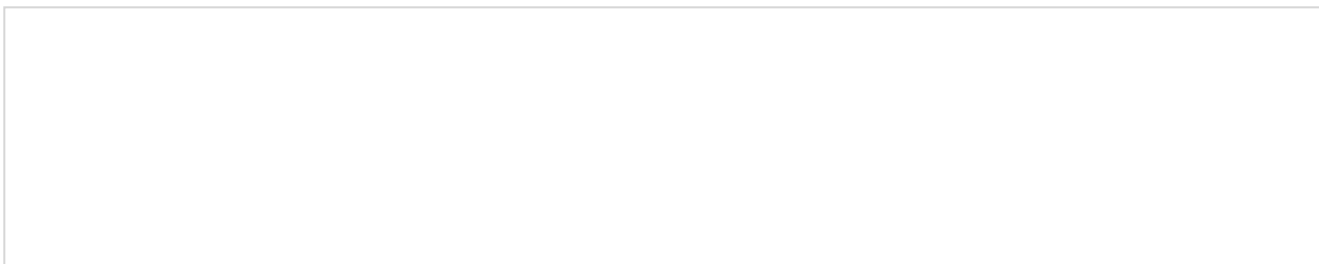
CSS3 Repeat Radial Gradients

[Live Demo](#)

```
<html>
  <head>
    <style>
      #grad1 {
        height: 100px;
        width: 550px;
        background: -webkit-repeating-radial-gradient(blue, yellow 10%, green 15%);
        background: -o-repeating-radial-gradient(blue, yellow 10%, green 15%);
        background: -moz-repeating-radial-gradient(blue, yellow 10%, green 15%);
        background: repeating-radial-gradient(blue, yellow 10%, green 15%);
      }
    </style>
  </head>

  <body>
    <div id = "grad1"></div>
  </body>
</html>
```

It will produce the following result –



CSS3 - Shadow

```
<html>
  <head>
    <style>
      h1 {
        text-shadow: 2px 2px;
      }
      h2 {
```



```
text-shadow: 2px 2px 3px red;
}
h4 {
  color: white;
  text-shadow: 2px 2px 4px #000000;
}
h5 {
  text-shadow: 0 0 3px #FF0000;
}
h6 {
  text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;
}
p {
  color: white;
  text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
}
</style>
</head>

<body>
  <h1>Tutorialspoint.com</h1>
  <h2>Tutorialspoint.com</h2>
  <h3>Tutorialspoint.com</h3>
  <h4>Tutorialspoint.com</h4>
  <h5>Tutorialspoint.com</h5>
  <h6>Tutorialspoint.com</h6>
  <p>Tutorialspoint.com</p>
</body>
</html>
```

It will produce the following result –



Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

Tutorialspoint.com

box shadow

Used to add shadow effects to elements, Following is the example to add shadow effects to element.

Live Demo

```
<html>
  <head>
    <style>
      div {
        width: 300px;
        height: 100px;
        padding: 15px;
        background-color: red;
        box-shadow: 10px 10px;
      }
    </style>
  </head>

  <body>
    <div>This is a div element with a box-shadow</div>
  </body>
</html>
```

It will produce the following result –



CSS3 - Text

Sr.No. Value & Description 1

text-align-last

Used to align the last line of the text

2

text-emphasis

Used to emphasis text and color

3

text-overflow

used to determines how overflowed content that is not displayed is signaled to users

4

word-break

Used to break the line based on word

5

word-wrap

Used to break the line and wrap onto next line

Text-overflow

The text-overflow property determines how overflowed content that is not displayed is signaled to users. the sample example of text overflow is shown as follows –

Live Demo

```
<html>
  <head>
    <style>
      p.text1 {
        white-space: nowrap;
        width: 500px;
```



```
p.text2 {  
    white-space: nowrap;  
    width: 500px;  
    border: 1px solid #000000;  
    overflow: hidden;  
    text-overflow: ellipsis;  
}  
</style>  
</head>  
  
<body>  
  
    <b>Original Text:</b>  
  
    <p>  
        Tutorials Point originated from the idea that there exists a class of  
        readers who respond better to online content and prefer to learn new  
        skills at their own pace from the comforts of their drawing rooms.  
    </p>  
  
    <b>Text overflow:clip:</b>  
  
    <p class = "text1">  
        Tutorials Point originated from the idea that there exists  
        a class of readers who respond better to online content and prefer  
        to learn new skills at their own pace from the comforts of their  
        drawing rooms.  
    </p>  
  
    <b>Text overflow:ellipsis</b>  
  
    <p class = "text2">  
        Tutorials Point originated from the idea that there exists  
        a class of readers who respond better to online content and  
        prefer to learn new skills at their own pace from the comforts  
        of their drawing rooms.  
    </p>  
  
</body>  
</html>
```

It will produce the following result –



online content and prefer to learn new skills at their own pace from the comforts of their drawing rooms.

Text overflow:clip

Tutorials Point originated from the idea that there exists a class of readers who

Text overflow:ellipsis

Tutorials Point originated from the idea that there exists a class of readers ...

CSS3 Word Breaking

Used to break the line, following code shows the sample code of word breaking.

[Live Demo](#)

```
<html>
  <head>
    <style>
      p.text1 {
        width: 140px;
        border: 1px solid #000000;
        word-break: keep-all;
      }
      p.text2 {
        width: 140px;
        border: 1px solid #000000;
        word-break: break-all;
      }
    </style>
  </head>

  <body>

    <b>line break at hyphens:</b>
    <p class = "text1">
      Tutorials Point originated from the idea that there exists a
      class of readers who respond better to online content and prefer
      to learn new skills at their own pace from the comforts of
      their drawing rooms.
    </p>

    <b>line break at any character</b>

    <p class = "text2">
      Tutorials Point originated from the idea that there exists a
```



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

It will produce the following result –

line break at hyphens:

Tutorials Point
originated from the
idea that there exists-
a class of readers
who respond better to
online content and
prefer to learn new
skills at their own
pace from the
comforts of their
drawing rooms.

line break at any character

Tutorials Point origin
ated from the idea tha
t there exists a class o
f readers who respon
d better to online con
tent and prefer to lear
n new skills at their o
wn pace from the co
mforts of their drawi
ng rooms.

CSS word wrapping

Word wrapping is used to break the line and wrap onto next line. the following code will have sample syntax –

```
p {  
  word-wrap: break-word;  
}
```

CSS3 - Web Fonts



TrueType is an outline font standard developed by Apple and Microsoft in the late 1980s, it became most common fonts for both windows and MAC operating systems.

2

OpenType Fonts (OTF)

OpenType is a format for scalable computer fonts and developed by Microsoft

3

The Web Open Font Format (WOFF)

WOFF is used for develop web page and developed in the year of 2009. Now it is using by W3C recommendation.

4

SVG Fonts/Shapes

SVG allow SVG fonts within SVG documentation. We can also apply CSS to SVG with font face property.

5

Embedded OpenType Fonts (EOT)

EOT is used to develop the web pages and it has embedded in webpages so no need to allow 3rd party fonts

Following code shows the sample code of font face –

Live Demo

```
<html>
  <head>
    <style>
      @font-face {
        font-family: myFirstFont;
        src: url(/css/font/SansationLight.woff);
      }
      div {
        font-family: myFirstFont;
      }
    </Style>
  </head>

  <body>
    <div>This is the example of font face with CSS3.</div>
    <p><b>Original Text :</b>This is the example of font face with CSS3.</p>
  </body>
</html>
```

It will produce the following result –



Fonts description

The following list contained all the fonts description which are placed in the @font-face rule –

Sr.No.	Value & Description
1	font-family Used to defines the name of font
2	src Used to defines the URL
3	font-stretch Used to find, how font should be stretched
4	font-style Used to defines the fonts style
5	font-weight Used to defines the font weight(boldness)

CSS3 - 2d Transforms

1

matrix(n,n,n,n,n,n)

Used to defines matrix transforms with six values

2

translate(x,y)

Used to transforms the element along with x-axis and y-axis

3



4

translateY(n)

Used to transform the element along with y-axis

5

scale(x,y)

Used to change the width and height of element

6

scaleX(n)

Used to change the width of element

7

scaleY(n)

Used to change the height of element

8

rotate(angle)

Used to rotate the element based on an angle

9

skewX(angle)

Used to define skew transforms along with x axis

10

skewY(angle)

Used to define skew transforms along with y axis

The following examples are shown the sample of all above properties.

Rotate 20 degrees

Box rotation with 20 degrees angle as shown below –

[Live Demo](#)

```
<html>
  <head>
    <style>
      div {
        width: 300px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
```



```
ms-transform: rotate(20deg);

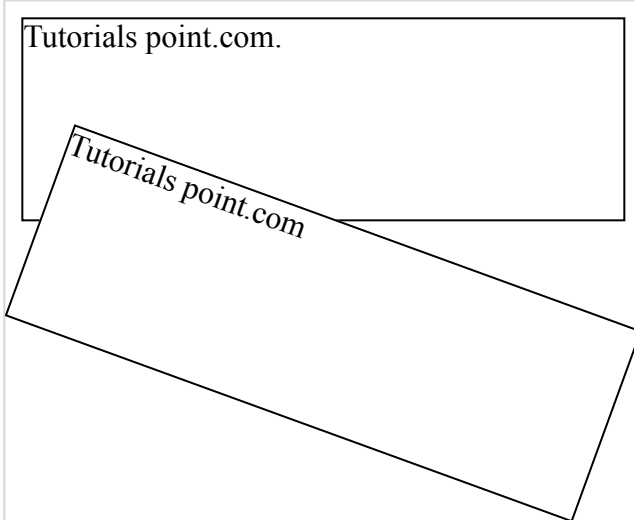
/* Safari */
-webkit-transform: rotate(20deg);

/* Standard syntax */
transform: rotate(20deg);
}
</style>
</head>

<body>
  <div>
    Tutorialspoint.com.
  </div>

  <div id = "myDiv">
    Tutorialspoint.com
  </div>
</body>
</html>
```

It will produce the following result –



Rotate -20 degrees

Box rotation with -20 degrees angle as shown below –

```
<html>
  <head>
    <style>
```

Live Demo



```
background-color: pink;
border: 1px solid black;
}
div#myDiv {
  /* IE 9 */
  -ms-transform: rotate(-20deg);

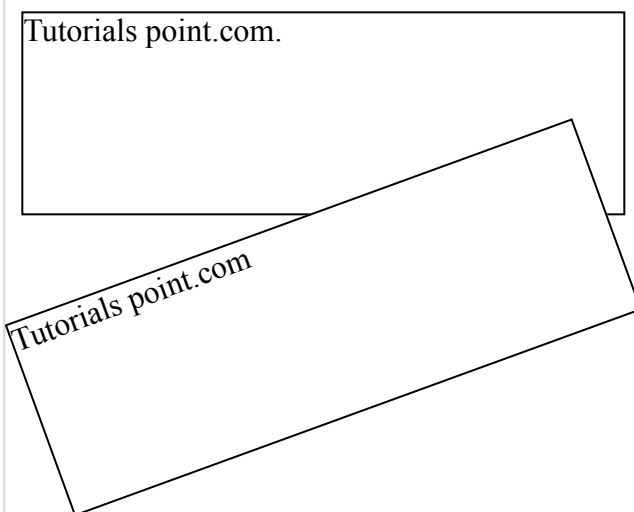
  /* Safari */
  -webkit-transform: rotate(-20deg);

  /* Standard syntax */
  transform: rotate(-20deg);
}
</style>
</head>

<body>
  <div>
    Tutorials point.com.
  </div>

  <div id = "myDiv">
    Tutorials point.com
  </div>
</body>
</html>
```

It will produce the following result –



Skew X axis

Box rotation with skew x-axis as shown below –



```
<style>
  div {
    width: 300px;
    height: 100px;
    background-color: pink;
    border: 1px solid black;
  }
  div#skewDiv {
    /* IE 9 */
    -ms-transform: skewX(20deg);

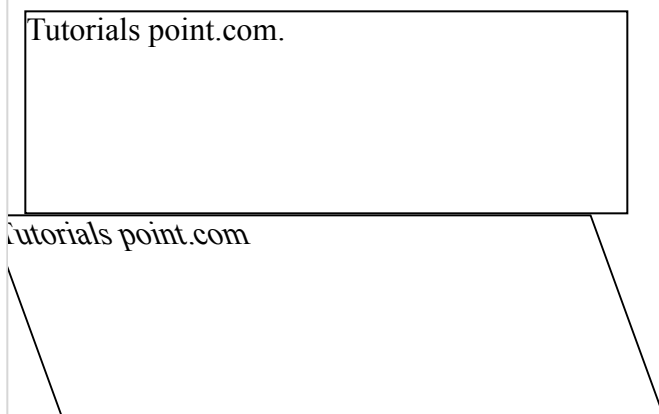
    /* Safari */
    -webkit-transform: skewX(20deg);

    /* Standard syntax */
    transform: skewX(20deg);
  }
</style>
</head>

<body>
  <div>
    Tutorialspoint.com.
  </div>

  <div id = "skewDiv">
    Tutorialspoint.com
  </div>
</body>
</html>
```

It will produce the following result –





Live Demo

```
<html>
  <head>
    <style>
      div {
        width: 300px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
      }
      div#skewDiv {
        /* IE 9 */
        -ms-transform: skewY(20deg);

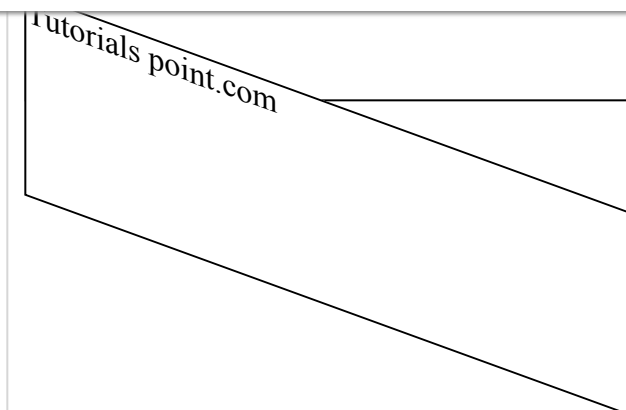
        /* Safari */
        -webkit-transform: skewY(20deg);

        /* Standard syntax */
        transform: skewY(20deg);
      }
    </style>
  </head>

  <body>
    <div>
      Tutorials point.com.
    </div>

    <div id = "skewDiv">
      Tutorials point.com
    </div>
  </body>
</html>
```

It will produce the following result –



Matrix transform

Box rotation with Matrix transforms as shown below –

[Live Demo](#)

```
<html>
  <head>
    <style>
      div {
        width: 300px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
      }
      div#myDiv1 {
        /* IE 9 */
        -ms-transform: matrix(1, -0.3, 0, 1, 0, 0);

        /* Safari */
        -webkit-transform: matrix(1, -0.3, 0, 1, 0, 0);

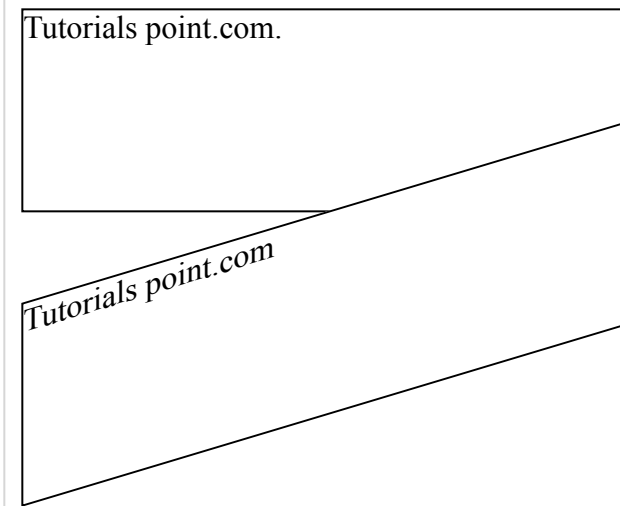
        /* Standard syntax */
        transform: matrix(1, -0.3, 0, 1, 0, 0);
      }
    </style>
  </head>

  <body>
    <div>
      Tutorialspoint.com.
    </div>

    <div id = "myDiv1">
      Tutorialspoint.com
    </div>
  </body>
</html>
```



It will produce the following result –



Matrix transforms with another direction.

[Live Demo](#)

```
<html>
  <head>
    <style>
      div {
        width: 300px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
      }
      div#myDiv2 {
        /* IE 9 */
        -ms-transform: matrix(1, 0, 0.5, 1, 150, 0);

        /* Safari */
        -webkit-transform: matrix(1, 0, 0.5, 1, 150, 0);

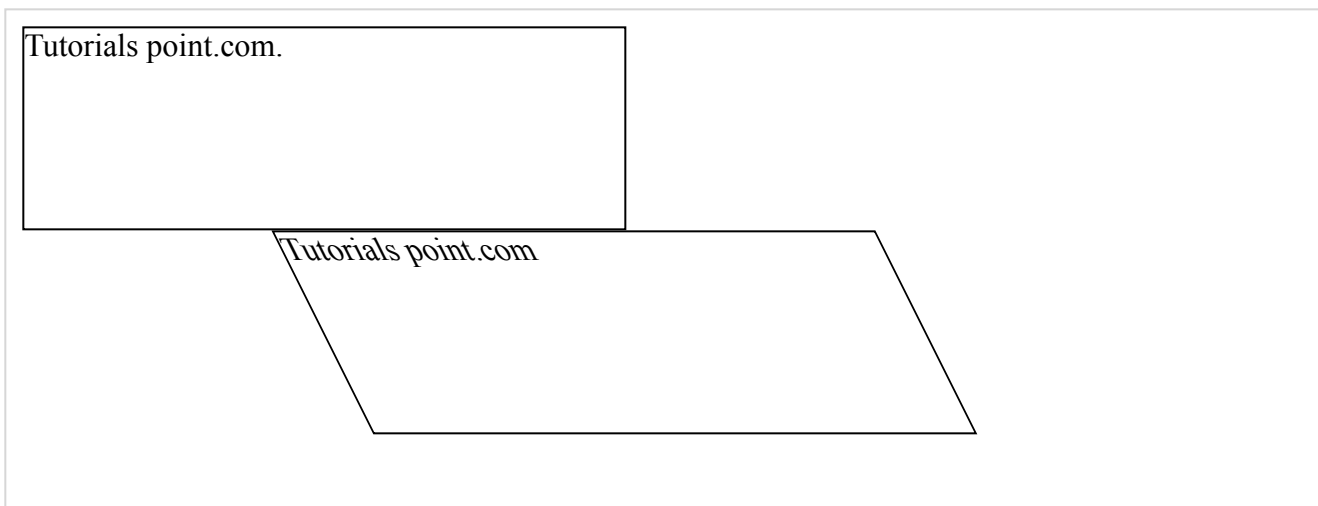
        /* Standard syntax */
        transform: matrix(1, 0, 0.5, 1, 150, 0);
      }
    </style>
  </head>

  <body>
    <div>
      Tutorials point.com.
    </div>
  </body>
</html>
```



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

It will produce the following result –



CSS3 - 3D Transforms

1

matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n)

Used to transform the element by using 16 values of matrix

2

translate3d(x,y,z)

Used to transform the element by using x-axis, y-axis and z-axis

3

translateX(x)

Used to transform the element by using x-axis

4

translateY(y)

Used to transform the element by using y-axis

5

translateZ(z)

Used to transform the element by using z-axis

6



scaleY(y)

Used to scale transforms the element by using y-axis

8

scaleY(y)

Used to transforms the element by using z-axis

9

rotateX(angle)

Used to rotate transforms the element by using x-axis

10

rotateY(angle)

Used to rotate transforms the element by using y-axis

11

rotateZ(angle)

Used to rotate transforms the element by using z-axis

X-axis 3D transforms

The following an example shows the x-axis 3D transforms.

Live Demo

```
<html>
  <head>
    <style>
      div {
        width: 200px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
      }
      div#myDiv {
        -webkit-transform: rotateX(150deg);

        /* Safari */
        transform: rotateX(150deg);

        /* Standard syntax */
      }
    </style>
  </head>
```



```

</div>
    tutorials point.com
</div>

<p>Rotate X-axis</p>

<div id = "myDiv">
    tutorials point.com.
</div>

</body>
</html>

```

It will produce the following result –

tutorials point.com

Rotate X-axis

tutorials point.com

Y-axis 3D transforms

The following an example shows the y-axis 3D transforms –

```

<html>
<head>
<style>
    div {
        width: 200px;
        height: 100px;
        background-color: pink;
        border: 1px solid black;
    }
    div#yDiv {
        -webkit-transform: rotateY(150deg);

```

Live Demo



```

        /* Standard syntax */
    }
</style>
</head>

<body>

    <div>
        tutorials point.com
    </div>

    <p>Rotate Y axis</p>

    <div id = "yDiv">
        tutorials point.com.
    </div>

</body>
</html>

```

It will produce the following result –

tutorials point.com

Rotate Y axis

tutorials point.com.

Z-axis 3D transforms

The following an example shows the Z-axis 3D transforms –

```

<html>
  <head>
    <style>

```

Live Demo



```
background-color: pink;
border: 1px solid black;
}
div#zDiv {
  -webkit-transform: rotateZ(90deg);

  /* Safari */
  transform: rotateZ(90deg);

  /* Standard syntax */
}
</style>
</head>

<body>
  <div>
    tutorials point.com
  </div>

  <p>rotate Z axis</p>

  <div id = "zDiv">
    tutorials point.com.
  </div>
</body>
</html>
```

It will produce the following result –



CSS3 - Animation

```
} div { width: 100px; height: 100px; background-color: red; animation-name: animation; animation-duration: 5s; }
```

The above example shows height, width, color, name and duration of animation with keyframes syntax.

Moving left animation

[Live Demo](#)

```
<html>
  <head>
    <style type = "text/css">
      h1 {
        -moz-animation-duration: 3s;
        -webkit-animation-duration: 3s;
        -moz-animation-name: slidein;
        -webkit-animation-name: slidein;
      }
      @-moz-keyframes slidein {
        from {
          margin-left:100%;
          width:300%
        }
        to {
          margin-left:0%;
          width:100%;
        }
      }
    </style>
  </head>
  <body>
    <div>
      <h1>
        </h1>
      </div>
    </body>
  </html>
```



```

        margin-left:100%;
        width:300%
    }
    to {
        margin-left:0%;
        width:100%;
    }
}
</style>
</head>

<body>
    <h1>Tutorials Point</h1>
    <p>this is an example of moving left animation .</p>
    <button onclick = "myFunction()">Reload page</button>
    <script>
        function myFunction() {
            location.reload();
        }
    </script>
</body>
</html>

```

It will produce the following result –

Tutorials Point

this is an example of moving left animation .

Reload page

Moving left animation with keyframes

Live Demo

```

<html>
<head>
    <style type = "text/css">
        h1 {
            -moz-animation-duration: 3s;
            -webkit-animation-duration: 3s;
            -moz-animation-name: slidein;
            -webkit-animation-name: slidein;
        }
        @-moz-keyframes slidein {

```



```

    }
    75% {
        font-size:300%;
        margin-left:25%;
        width:150%;
    }
    to {
        margin-left:0%;
        width:100%;
    }
}
@-webkit-keyframes slidein {
    from {
        margin-left:100%;
        width:300%
    }
    75% {
        font-size:300%;
        margin-left:25%;
        width:150%;
    }
    to {
        margin-left:0%;
        width:100%;
    }
}
</style>
</head>

<body>
    <h1>Tutorials Point</h1>

    <p>This is an example of animation left with an extra keyframe
        to make text changes.</p>
    <button onclick = "myFunction()">Reload page</button>
    <script>
        function myFunction() {
            location.reload();
        }
    </script>
</body>
</html>

```

It will produce the following result –



This is an example of animation left with an extra keyframe to make text changes.

Reload page

CSS3 - Multi Columns

1

column-count

Used to count the number of columns that element should be divided.

2

column-fill

Used to decide, how to fill the columns.

3

column-gap

Used to decide the gap between the columns.

4

column-rule

Used to specifies the number of rules.

5

rule-color

Used to specifies the column rule color.

6

rule-style

Used to specifies the style rule for column.

7

rule-width

Used to specifies the width.

8

column-span

Used to specifies the span between columns.

Example



```

<html>
<head>
  <style>
    .multi {
      /* Column count property */
      -webkit-column-count: 4;
      -moz-column-count: 4;
      column-count: 4;

      /* Column gap property */
      -webkit-column-gap: 40px;
      -moz-column-gap: 40px;
      column-gap: 40px;

      /* Column style property */
      -webkit-column-rule-style: solid;
      -moz-column-rule-style: solid;
      column-rule-style: solid;
    }
  </style>
</head>

<body>

  <div class = "multi">
    Tutorials Point originated from the idea that there exists a class
    of readers who respond better to online content and prefer to learn
    new skills at their own pace from the comforts of their drawing rooms.
    The journey commenced with a single tutorial on HTML in 2006 and elated
    by the response it generated, we worked our way to adding fresh tutorials
    to our repository which now proudly flaunts a wealth of tutorials and
    allied articles on topics ranging from programming languages to web
    designing to academics and much more.
  </div>

</body>
</html>

```

It will produce the following result –



exists a class of readers who respond better to online content and prefer to learn new	drawing rooms. The journey commenced with a single tutorial on HTML in 2006 and elated	adding fresh tutorials to our repository which now proudly flaunts a wealth of tutorials	languages to web designing to academics and much more.
--	--	--	--

For suppose, if user wants to make text as new paper without line, we can do this by removing style syntax as shown below –

```
.multi {
  /* Column count property */
  -webkit-column-count: 4;
  -moz-column-count: 4;
  column-count: 4;

  /* Column gap property */
  -webkit-column-gap: 40px;
  -moz-column-gap: 40px;
  column-gap: 40px;
}
```

It will produce the following result –

Tutorials Point originated from the idea that there exists a class of readers who respond better to online content and prefer to learn new	skills at their own pace from the comforts of their drawing rooms. The journey commenced with a single tutorial on HTML in 2006 and elated	by the response it generated, we worked our way to adding fresh tutorials to our repository which now proudly flaunts a wealth of tutorials	and allied articles on topics ranging from programming languages to web designing to academics and much more.
--	--	---	---

CSS3 - User Interface

1

appearance

Used to allow the user to make elements as user interface elements.

2

box-sizing



icon

Used to provide the icon on area.

4

resize

Used to resize elements which are on area.

5

outline-offset

Used to draw the behind the outline.

6

nav-down

Used to move down when you have pressed on down arrow button in keypad.

7

nav-left

Used to move left when you have pressed on left arrow button in keypad.

8

nav-right

Used to move right when you have pressed on right arrow button in keypad.

9

nav-up

Used to move up when you have pressed on up arrow button in keypad.

Example of resize property

Resize property is having three common values as shown below –

horizontal

vertical

both

Using of **both** value in resize property in css3 user interface –

```
<html>
<head>
<style>
div {
border: 2px solid;
padding: 20px;
```

Live Demo



```
</style>
</head>

<body>
  <div>Tutorialspoint.com</div>
</body>
</html>
```

It will produce the following result –



Tutorialspoint.com

CSS3 Outline offset

Out line means draw a line around the element at outside of border.

[Live Demo](#)

```
<html>
  <head>
    <style>
      div {
        margin: 20px;
        padding: 10px;
        width: 300px;
        height: 100px;
        border: 5px solid pink;
        outline: 5px solid green;
        outline-offset: 15px;
      }
    </style>
  </head>

  <body>
    <div>Tutorialspoint</div>
  </body>
</html>
```

It will produce the following result –

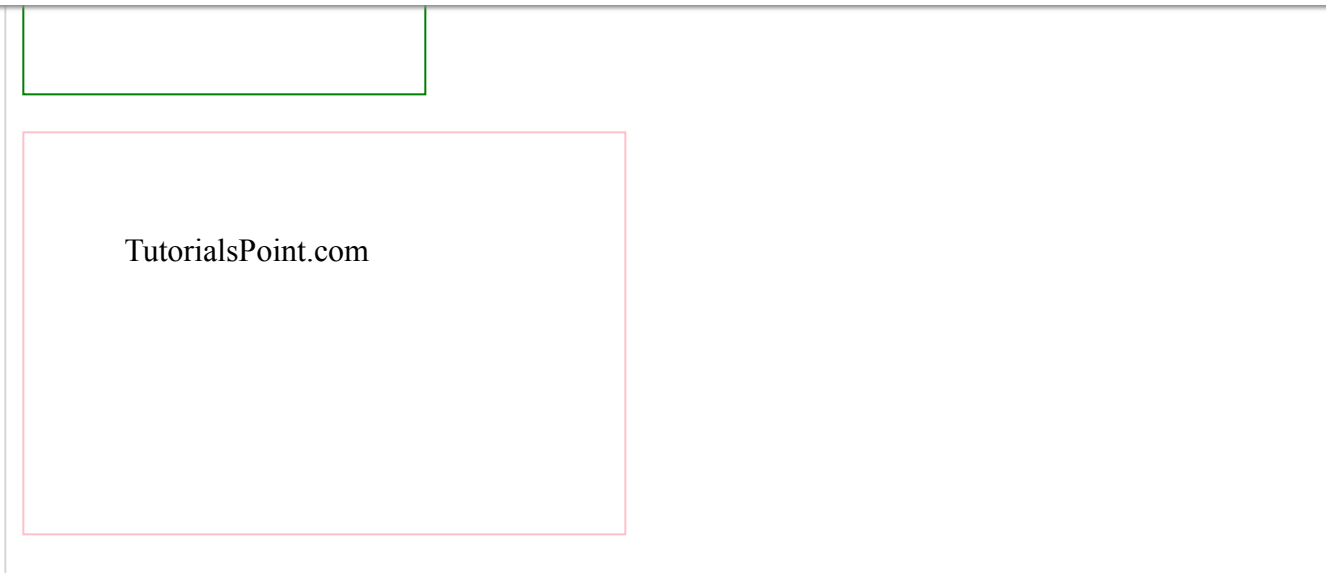


CSS3 - Box Sizing

```
<html>
  <head>
    <style>
      .div1 {
        width: 200px;
        height: 100px;
        border: 1px solid green;
      }
      .div2 {
        width: 200px;
        height: 100px;
        padding: 50px;
        border: 1px solid pink;
      }
    </style>
  </head>

  <body>
    <div class = "div1">TutorialsPoint.com</div><br />
    <div class = "div2">TutorialsPoint.com</div>
  </body>
</html>
```

It will produce the following result –



Above image is having same width and height of two element but giving result is different, cause second one is included padding property.

CSS3 box sizing property

[Live Demo](#)

```
<html>
  <head>
    <style>
      .div1 {
        width: 300px;
        height: 100px;
        border: 1px solid blue;
        box-sizing: border-box;
      }
      .div2 {
        width: 300px;
        height: 100px;
        padding: 50px;
        border: 1px solid red;
        box-sizing: border-box;
      }
    </style>
  </head>

  <body>
    <div class = "div1">TutorialsPoint.com</div><br />
    <div class = "div2">TutorialsPoint.com</div>
  </body>
</html>
```



TutorialsPoint.com

TutorialsPoint.com

Above elements are having same height and width with `box-sizing: border-box` so result is always same for both elements as shown above.

CSS - Responsive

```
<html> <head> <style> body { font: 600 14px/24px "Open Sans", "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lucida Grande", Sans-Serif; } h1 { color: #9799a7; font-size: 14px; font-weight: bold; margin-bottom: 6px; } .container:before, .container:after { content: ""; display: table; } .container:after { clear: both; } .container { background: #eaeaed; margin-bottom: 24px; *zoom: 1; } .container-75 { width: 75%; } .container-50 { margin-bottom: 0; width: 50%; } .container, section, aside { border-radius: 6px; } section, aside { background: #2db34a; color: #fff; margin: 1.858736059%; padding: 20px 0; text-align: center; } section { float: left; width: 63.197026%; } aside { float: right; width: 29.3680297%; } </style> </head> <body> <h1>100% Wide Container</h1> <div class = "container"> <section>Section</section> <aside>Aside</aside> </div> <h1>75% Wide Container</h1> <div class = "container container-75"> <section>Section</section> <aside>Aside</aside> </div> <h1>50% Wide Container</h1> <div class = "container container-50"> <section>Section</section> <aside>Aside</aside> </div> </body> </html>
```

It will produce the following result –



Media queries

Media queries is for different style rules for different size devices such as mobiles, desktops, etc.,

Live Demo

```
<html>
  <head>
    <style>
      body {
        background-color: lightpink;
      }
      @media screen and (max-width: 420px) {
        body {
          background-color: lightblue;
        }
      }
    </style>
  </head>

  <body>
    <p>
      If screen size is less than 420px, then it will show lightblue
      color, or else it will show light pink color
    </p>
  </body>
</html>
```

It will produce the following result –



Bootstrap responsive web design

Bootstrap is most popular web design framework based on HTML, CSS and Java script and it helps you to design web pages in responsive way for all devices.

[Live Demo](#)

```
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport" content = "width=device-width, initial-scale = 1">
    <link rel = "stylesheet"
      href = "http://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <style>
      body {
        color: green;
      }
    </style>
  </head>

  <body>

    <div class = "container">

      <div class = "jumbotron">
        <h1>Tutorials point</h1>
        <p>
          Tutorials Point originated from the idea that there exists a class
          of readers who respond better to online content and prefer to learn
          new skills at their own pace from the comforts of their drawing rooms.
        </p>
      </div>

      <div class = "row">
        <div class = "col-md-4">
          <h2>Android</h2>
          <p>
            Android is an open source and Linux-based operating system for mobile
            devices such as smartphones and tablet computers. Android was developed
            by the Open Handset Alliance, led by Google, and other companies.
          </p>
        </div>

        <div class = "col-md-4">
          <h2>CSS</h2>
        </div>
      </div>
    </div>
  </body>
</html>
```



```
</p>
</div>

<div class = "col-md-4">
  <h2>Java</h2>
  <p>
    Java is a high-level programming language originally developed by Sun
    Microsystems and released in 1995. Java runs on a variety of platforms,
    such as Windows, Mac OS, and the various versions of UNIX. This tutorial
    gives a complete understanding of Java.
  </p>
</div>
</div>

</body>
</html>
```

It will produce the following result –



tutorialspoint

Tutorials Point originated from the idea that there exists a class of readers who respond better to online content and prefer to learn new skills at their own pace from the comforts of their drawing rooms.

Android

Android is an open source and Linux-based operating system for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies.

CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

Java

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This tutorial gives a complete understanding of Java.

[Previous Page](#)[Next Page](#)

Advertisements

[About us](#)[Terms of use](#)[Cookies Policy](#)[FAQ's](#)[Helping](#)



Chapter 10

Document Object Model and Dynamic HTML

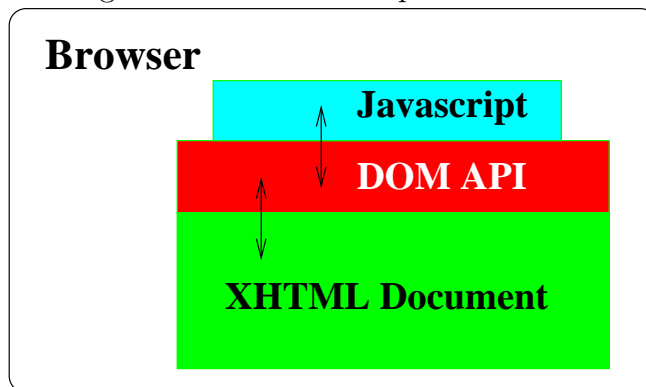
The term *Dynamic HTML*, often abbreviated as DHTML, refers to the technique of making Web pages dynamic by client-side scripting to manipulate the document content and presentation. Web pages can be made more lively, dynamic, or interactive by DHTML techniques.

With DHTML you can prescribe actions triggered by browser events to make the page more lively and responsive. Such actions may alter the content and appearance of any parts of the page. The changes are fast and efficient because they are made by the browser without having to network with any servers. Typically the client-side scripting is written in Javascript which is being standardized. Chapter 9 already introduced Javascript and basic techniques for making Web pages dynamic.

Contrary to what the name may suggest, DHTML is not a markup language or a software tool. It is a technique to make dynamic Web pages via client-side programming. In the past, DHTML relies on browser/vendor specific features to work. Making such pages work for all browsers requires much effort, testing, and unnecessarily long programs.

Standardization efforts at W3C and elsewhere are making it possible to write *standard-based DHTML* that work for all compliant browsers. Standard-based DHTML involves three aspects:

Figure 10.1: DOM Compliant Browser



1. Javascript—for cross-browser scripting (Chapter 9)
2. Cascading Style Sheets (CSS)—for style and presentation control (Chapter 6)
3. *Document Object Model* (DOM)—for a uniform programming interface to access and manipulate the Web page as a document

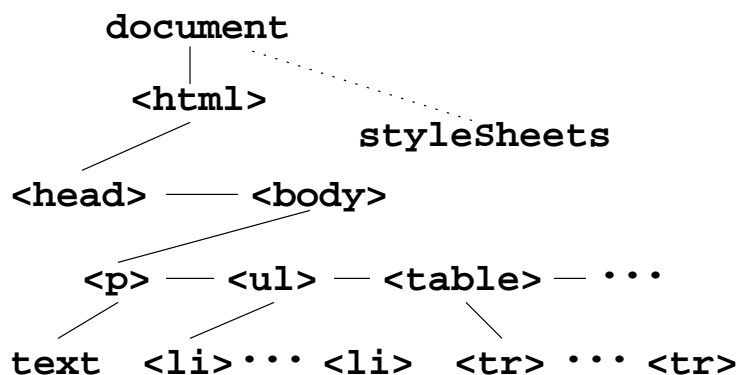
When these three aspects are combined, you get the ability to program changes in Web pages in reaction to user or browser generated events, and therefore to make HTML pages more dynamic.

Popular with Web developers, supported by all major browsers, and standardized, Javascript provides the ability to program browser actions in response to events. To have true cross-platform DHTML, we still need a uniform way for Javascript to access and manipulate Web documents. This brings us to the DOM.

10.1 What Is DOM?

With cooperation from major browser vendors, the W3C is establishing the *Document Object Model* (DOM) as a standard *application programming interface* (API) for scripts to access and manipulate HTML and XML documents. Compliant clients, including browsers and other user agents, provide the DOM specified API to access and modify the document being processed (Figure 10.1). The DOM API gives a logical view of the document where objects

Figure 10.2: DOM Tree Structure



represent different parts: windows, documents, elements, attributes, texts, events, style sheets, style rules, etc. These *DOM objects* are organized into a tree structure (the DOM tree) to reflect the natural organization of a document. HTML elements are represented by *tree nodes* and organized into a hierarchy. Each Web page has a **document** node at the root of the tree. The **head** and **body** nodes become *child nodes* of the **document** node (Figure 10.2).

From a node on the DOM tree, you can go down to any child node or go up to the parent node. With DOM, a script can add, modify, or delete elements and content by navigating the document structure, modifying or deleting existing nodes, and inserting dynamically built new nodes. Also attached to the document are its style sheets. Each element node on the DOM tree also contains a *style object* representing the display style for that element. Thus, through the DOM tree, style sheets and individual element styles can also be accessed and manipulated. Therefore, any parts of a page can be accessed, changed, deleted, or added and the script will work for any DOM compliant client.

DOM also specifies events available for page elements. As a result, most events that used to be reserved for links now work for all types of elements, giving the designer many more ways to make pages dynamic and responsive.

10.2 A Demonstration

Let's look at a simple example (Ex: **DomHello**) to illustrate DHTML. Figure 10.3 shows a very simple page with the phrase *Hello World Wide Web* on it. And Figure 10.4 shows that phrase becoming blue in a larger font on mouseover. The phrase goes back to normal again on mouseout. This is not an image rollover.

The HTML source for the page is

```
<head><title>Hello WWW with DOM</title>
<script type="text/javascript" src="hwww.js">
</script></head><body>
<p>Move the mouse over the phrase:</p>
<p><span id="hello" onmouseover="over()"
      onmouseout="out()">Hello World Wide Web</span>
---and see what happens.</p>
</body></html>
```

Note we have attached the `onmouseover` and `onmouseout` event handlers to part of a paragraph identified by the `span` with `id="hello"`.

The Javascript defined event handling functions are in the file `hwww.js`:

```
function over()
{
  el = document.getElementById("hello");    // (1)
  el.style.color = "blue";                  // (2)
  el.style.fontSize = "18pt";               // (3)
  el.style.fontWeight = "bold";             // (4)
}
```

Figure 10.3: Hello WWW

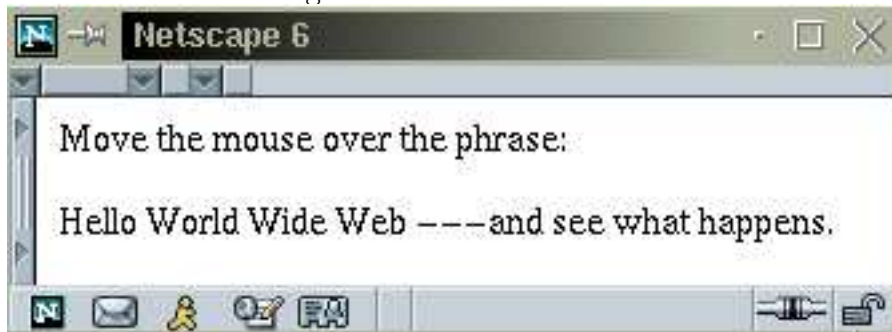
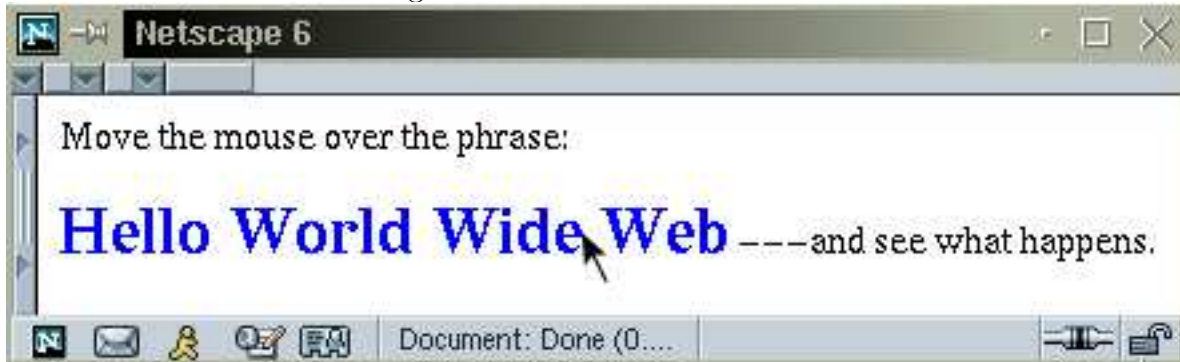


Figure 10.4: Mouse over Phrase



```
function out()  
{  
  el = document.getElementById("hello");  
  el.style.color = "";           // sets to default value  
  el.style.fontSize = "";  
  el.style.fontWeight = "";  
}
```

The `over` function obtains a reference to the target `` by using the `getElementById` method of the `document` element (line 1). The `document` is the root node of the DOM tree and offers many useful properties and methods. The convenient call

```
document.getElementById(str)
```

gives you the element with `str` as `id` (or `name`). It returns a reference to the node on the DOM tree that represents the desired element.

Once you have the node for a particular element, you can go to work on that node accessing information from it or making changes to it. Here the `over` function sets style properties for the element (lines 2-4) causing the `` to display in blue with 18-point boldface font (Figure 10.4). The `out` function sets these style properties to the empty string to go back to the default display (Figure 10.3).

10.3 DOM History and Architecture

Early browsers such as NN 3.0 and IE 3.0 have their own object models for representing documents. Starting as early as 1997, the W3C began to organize the DOM working group for establishing a cross-platform and language-neutral standard for access, traversal, and manipulation of document objects. The first W3C recommendation was DOM Level 1 completed in October 1998. DOM Level 1 specifies a *standard object-oriented interface* to HTML and XML documents. The *Level 1 Core* specifies the most central interfaces for the DOM tree. The DOM Level 1 HTML and XML specifications inherit from the Core and specialize in HTML and XML documents, respectively. The DOM specification for HTML/XHTML is most important for website development. The very first DOM specification, informally referred to as DOM Level 0, was built on existing conventions and practices supported by NN 3.0 and IE 3.0. A second edition of DOM Level 1 is being finalized.

In November 2000, DOM Level 2 was completed and it extended Level 1 by adding support for XML 1.0 namespaces, CSS, events and event handling for user interfaces and for tree manipulation, and tree traversal. The Level 2 HTML specification was becoming a W3C recommendation in 2002. DOM Level 3, still being developed, will add more sophisticated XML support, the ability to load and save documents, etc.

As DOM evolves through levels of enhancements, its basic architecture remains stable. The DOM architecture consists of *modules* covering different domains of the document object model:

- DOM Core—specifies the DOM tree, tree nodes, its access, traversal, and manipulation. The DOM Range and DOM Traversal modules provide higher-level methods for manipulating the DOM tree defined by the Core.
- DOM HTML—inherits from the Core and provides specialized and convenient ways to access and manipulate HTML/XHTML documents.
- DOM XML—inherits from the Core and provides support for XML specific needs.

- DOM Events—specifies events and event handling for user interfaces and the DOM tree. With DOM Events, drag and drop programs, for example, can be standardized.
- DOM CSS—defines easy to use ways to manipulate Cascading Style Sheets for the formatting and presentation of documents.

There are other modules and they can be found at the W3C site: www.w3.org/DOM/.

When using Javascript to write DOM related code, it is important to realize that not everything has been standardized. In particular, the `window` object is very browser dependent. Also certain fields such as `element.innerHTML` and `document.location`, are not part of the DOM specification.

10.4 Browser Support of DOM

Major vendors realize the importance of DOM and have begun to make their Web browsers DOM compliant. NN 7 and IE 6 already have good DOM support. In particular NN led the way in supporting DOM Level 1 and Level 2. Most examples in this chapter will work under both NN 6, IE 6 and later versions.

To detect the extent of DOM support that a user agent (browser) provides, the following type of Javascript code can be used:

```
var imp = document.implementation;
if ( typeof imp != "undefined" &&
    imp.hasFeature("HTML", "1.0") &&
    imp.hasFeature("Events", "2.0") &&
    imp.hasFeature("CSS", "2.0")
    )
{
    . . .
}
```

A browser is DOM compliant if it supports the interfaces specified by DOM. But it can also add interfaces not specified or add fields and methods to the required interfaces. For

example NN and IE both add `innerHTML` to the `HTMLElement` interface. It is easy to test if a field or method is available in a browser. For example,

```
if ( document.getElementById )  
    . . .
```

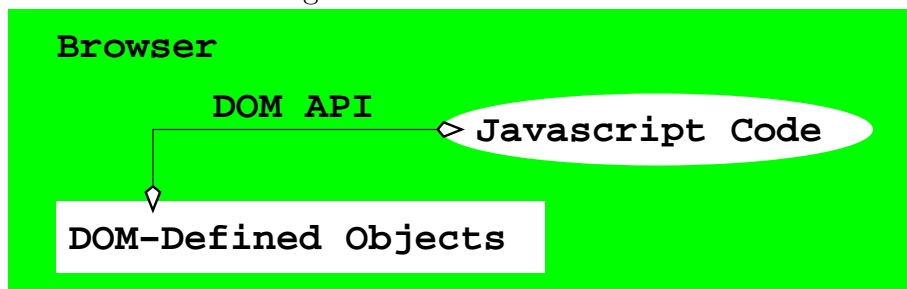
tests if the `getElementById` method is available in the `document` object.

DOM compliance test suites are available from www.w3.org/DOM/Test/.

10.5 DOM API Overview

DOM is a “platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.”

Figure 10.5: The DOM API



The DOM specifies an API (application programming interface) and provides a structural view of the document. DOM lists required interface objects and the *methods* (functions in the object) and *fields* (data entries in the object) each object must support. It is up to compliant browsers (agents) to supply concrete implementation, in a particular programming language and environment, for these objects, fields and methods. NN, IE and other browsers support DOM through standard Javascript (Figure 10.5).

As an interface, each DOM object *exposes* a set of fields and methods for Javascript to access and manipulate the underlying data structure that actually implements the document

structure. The situation is like a radio interface exposing a set of knobs and dials for the user. If the radio interface were standardized, then a robot would be able to operate any standard compliant radio.

The DOM tree represents the logical structure of a document. Each tree node is a **Node** object. There are different types of nodes that all *inherit* the basic **Node** interface. Inheritance is an important object-oriented programming (OOP) concept. In OOP, interfaces are organized into a hierarchy where *extended interfaces* inherit methods and properties required by **base interfaces**. The situation is quite like defining various upgraded car models by inheriting and adding to the features of a base model. In DOM, the **Node** object sits at the top of the interface hierarchy and many types of DOM tree nodes are directly or indirectly derived from **Node**. This means all DOM tree node types must support the properties and methods required by **Node**.

On the DOM tree, some types of nodes are *internal nodes* that may have *child nodes* of various types. *Leaf nodes*, on the other hand, have no child nodes. While DOM has many uses, our discussion focuses on *DOM HTML* which applies to HTML documents.

For any Web page, the root of the DOM tree is an **HTMLDocument** node and it is usually available directly from Javascript as **document** or **window.document**. The **document** object implements the **HTMLDocument** interface which gives you access to all the quantities associated with a Web page such as **URL**, **stylesheets**, **title**, **characterSet**, and many others (Section 10.14). The field **document.documentElement** gives you the child node, of type **HTMLElement**, that typically represents the **<html>** element (Figure 10.2). **HTMLElement** (Section 10.9) is the base interface for derived interfaces representing the many different HTML elements.

DOM Tree Nodes

The DOM tree for a Web page consists of different types of nodes (of type **Node**) including:

HTMLDocument —Root of the DOM tree providing access to page-wide quantities, stylesheets, markup elements, and, in most cases, the **<html>** element as a child node.

HTMLElement —Internal and certain leaf nodes on the DOM tree representing an HTML markup element. The **HTMLElement** interface provides access to element attributes and child nodes that may represent text and other HTML elements. Because we focus on the use of DOM in DHTML, we will use the terms *element* and *HTML element* interchangeably. The `document.getElementById(id)` call gives you any element with the given *id*.

Attr —An attribute in an **HTMLElement** object providing the ability to access and set an attribute. The **name** field (a string) of an **Attr** object is read-only while the **value** field can be set to a desired string. The **attributes** field of an **HTMLElement** object gives you a **NamedNodeMap** of **Attr** objects. Use the **length** property and the **item(index)** method of the named node map to visit each attribute. All DOM indices are zero-based.

Text —A leaf node containing the text inside a markup element. If there is no markup inside an element's content, the text is contained in a single **Text** object that is the only child of the element. The **wholeText** (or **data**) field returns the entire text as a string. Set the **data** string or call the **replaceWholeText(str)** method to make *str* the new text.

10.6 Getting Started with DOM

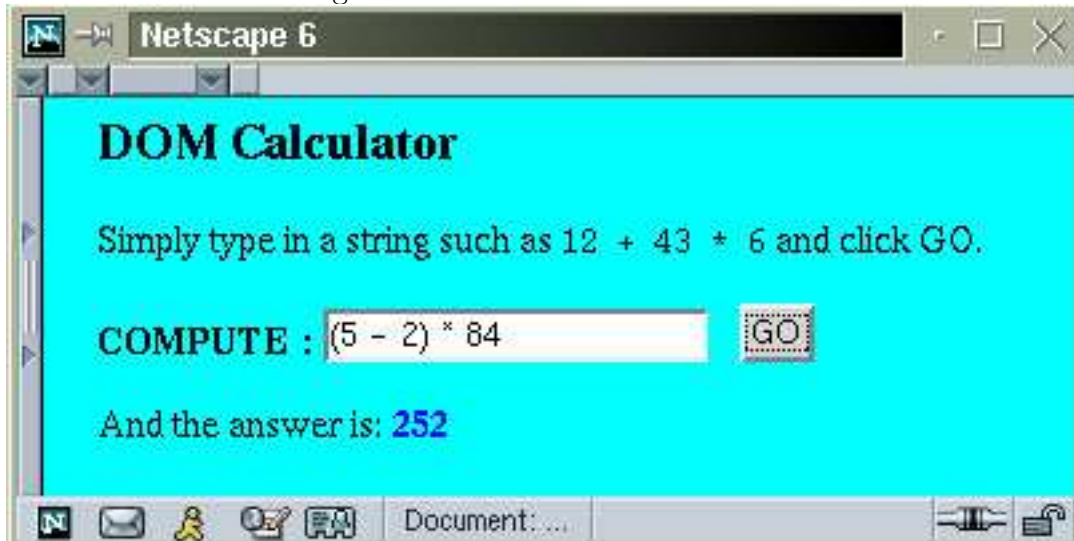
Let's create a simple calculator (Ex: **DomCalc**) to demonstrate DOM and DHTML. The user enters an arithmetic expression and clicks a button to perform the required computations. The answer is displayed in the regular running text of the page (Figure 10.6).

The HTML source shows the code for the input control (line A), the **GO** button (line B) and the **** for displaying the computed result (line C).

```
<head><title>DOM Calculator</title>
<link rel="stylesheet" href="domcalc.css"
      type="text/css" title="Dom Calculator" />
```

Brooks/Cole book/January 28, 2003

Figure 10.6: A DHTML Calculator



```

<script type="text/javascript" src="domcalc.js"></script>
</head>
<body onload="init()"> /* initialization onload */
<h3>DOM Calculator</h3>
<p>Simply type in a string such as
    <code>12 + 43 * 6</code> and click GO.</p>
<p><strong>COMPUTE : </strong>
    <input id="uin"
        value="(5 - 2) * 8" maxlength="30" />      (A)
    &nbsp;&nbsp;&nbsp;<input value="GO" type="button"
        onclick="comp('uin')" />                  (B)
</p><p id="par">And the answer is:
<span id="ans">00</span></p>                      (C)
</body>

```

The calculator is initialized immediately after page loading. The `init` and the `comp` (line B) event handlers are in the Javascript file `domcalc.js`:

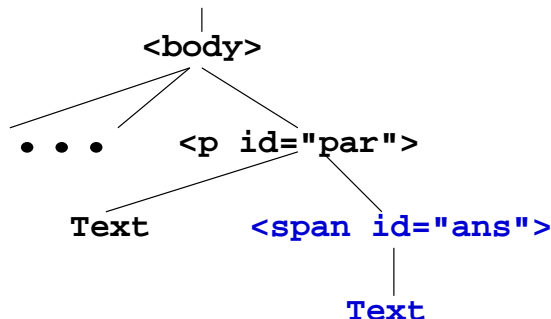
```

var answer;

function init()
{
    answer = document.getElementById("ans")
                .firstChild;                // (D)
    comp("uin");
}

```

Figure 10.7: Partial DOM Tree for Calculator Example



```

function comp(id)
{  var el = document.getElementById(id);    // (E)
  var res = eval(el.value);                // (F)
  answer.data = res;                       // (G)
}

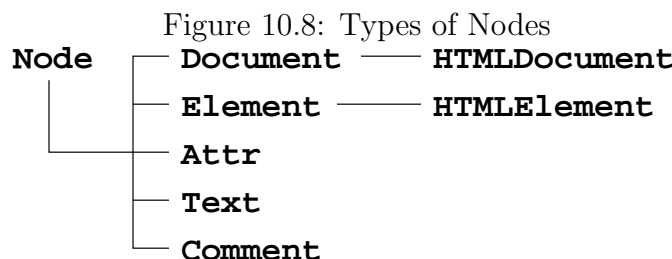
```

The global variable `answer` is initialized by the `init` function which is called via the `onload` event, an event that takes place immediately after page loading is complete. The variable `answer` holds the `Text` node, a lone child node in this case, of the `ans` `` (line D).

The `comp` function is called with the `id` of the user input element. The function obtains the input text as the `value` field of the input element `el` (line E), evaluates the input expression (line F), and sets the text of the `ans` `` to the result obtained by setting the `data` field of the `Text` node `answer` (line G).

Without DOM, Javascript computed results are usually placed in `<input>` or `<textarea>` elements (Ex: **Convert** in Section 9.14). Using the DOM interface, script computed results can be placed anywhere on a displayed page by modifying the DOM tree. Figure 10.7 shows the part of the DOM tree (in dark blue) that is used to display the results for the calculator.

`HTMLDocument` and `HTMLElement` interfaces are important and provide many methods and properties useful in practice. They inherit from the basic `Node` interface which is presented next.



10.7 The DOM Node Interface

In object-oriented programming, an interface specifies data values (called *fields*,¹) and functions (called *methods*) that are made available to application programs.

The **Node** interface is the base of all other node types on the DOM tree and provides useful fields and methods for them.

Node Fields

Fields provided by a **Node** are read-only and include:

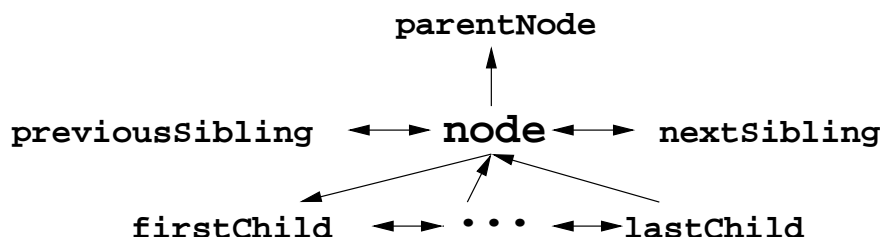
- **nodeType**—A small integer representing the *derived type* of the node. Figure 10.8 shows common derived node types. The **Node** interface provides symbolic constants, such as **ELEMENT_NODE** and **TEXT_NODE**, for values of **nodeType** (See WDP site for a list). The function **whichType** demonstrates how to determine node type (Ex: **WhichType**):

```

function whichType(nd)                                // (a)
{  if ( nd.nodeType == Node.ELEMENT_NODE )            // (b)
    window.alert("Element Node");
    else if ( nd.nodeType == Node.ATTRIBUTE_NODE )
        window.alert("Attribute Node");
    else if ( nd.nodeType == Node.TEXT_NODE )
        window.alert("Text Node");
    ...
}
```

¹In the official DOM specification, fields are called *attributes*. To distinguish them from HTML attributes, we use the commonly accepted term *fields* here.

Figure 10.9: Node Relations



The parameter `nd` is any DOM node whose type is to be determined (line a). The `nd.nodeType` is compared with the type constants defined by the `Node` interface to determine the node type of `nd` (line b).

- `parentNode`, `firstChild`, `lastChild`, `previousSibling`, and `nextSibling`—Related Nodes of a node (Figure 10.9).
- `nodeName` and `nodeValue`—Strings representing the name and value of a `Node`. The exact meaning of these strings depends on the node type, as shown in Table 10.1. For example the `nodeValue` of any `Element` or `HTMLElement` node is `null`.

Table 10.1: Meaning of `nodeName` and `nodeValue`

Node Type	nodeName	nodeValue
Element	Tag name	null
Attribute	Attribute name	Attribute value string
Text	#text	Text string
Entity	Entity name	null
Comment	#comment	Comment string

- `childNodes`—A `NodeList` of child nodes of the node. Some nodes have children and others don't. For a `Document` or an `HTMLElement` node, child nodes represent the HTML elements and text strings contained in that element.

The `length` field and the `item(i)` method of `NodeList` provide an easy way to visit each node on the node list. For example (Ex: `ChildNodes`), applying the function `visitChildren`:

```

function visitChildren(id)
{
  var nd = document.getElementById(id);
  var ch = nd.childNodes;
  var len = ch.length;          // number of nodes
  for ( i=0; i < len; i++)
  {
    nd = ch.item(i);           // node i
    window.alert( nd.nodeName + " "
                  + nd.nodeValue );
  }
}

```

on the element with id="par"

```
<p id="par">Here is <img ... /><br /> a picture.</p>
```

displays this sequence

```

#text    Here is
IMG
BR
#text    a picture.

```

- **attributes**—A `NamedNodeMap` of `Attr` nodes representing HTML attributes of the node. Attribute nodes are not child nodes of a node but are attached to the node via the `attributes` field. DOM defines a `NamedNodeMap` as a collection of nodes accessible by name. Thus, `attributes` is a list of `Attribute` objects representing the HTML attributes for a given node. Let `att = nd.attributes` be the attribute list of some node `nd`, then you can go through all listed attributes with the code (Ex: **AttrAccess**):

```

var len = att.length;
for ( i=0; i < len; i++ )
{
  window.alert(att.item(i).name + " = " +
               att.item(i).value );
}

```

The length of the attribute list `nd.attributes` can be browser dependent. NN lists only attributes set explicitly in the HTML code, whiel IE gives all possible attributes. To examine a specific attribute you can use, for example, the code

```
var b = att.getNamedItem("border");  
window.alert(b.value);           // value of border
```

The value returned by `getNamedItem` is a node with the given name in the `NamedNodeMap` or `null`.

The `ownerDocument` field of a node leads you to the root of the DOM tree. It is worth emphasizing that the fields of `Node` are read-only. Assignments to them have no effect.

Also, `NodeList` and `NamedNodeMap` objects in the DOM are *live*, meaning changes to the underlying document structure are reflected in all relevant `NodeList` and `NamedNodeMap` objects. For example, if you get the `childNodes` of an `HTMLElement`, then subsequently add or remove child nodes, the changes are automatically reflected in the `childNodes` you got before. This behavior is usually supported by returning a reference to the data structure containing the actual child nodes of the `HTMLElement`.

Node Methods

In addition to fields, the `Node` interface provides many methods, inherited by all node types. These fields and methods combine to provide the basis for accessing, navigating, and modifying the DOM tree. Specialized interfaces for other node types offer additional features for functionality and convenience.

Among `Node` methods, the following are more frequently used.

- `node.normalize()`—Adjusts the subtree rooted at *node* to remove empty nodes and to combine adjacent text nodes. The resulting *normalized DOM tree* has no empty or adjacent text nodes. Before normization, a DOM tree may contain empty and/or adjacent text nodes due to spacing and line breaks in the page source code. Such white space are often used to avoid long lines and to make the source easier to read. For example, the call

```
document.documentElement.normalize();
```

normalizes the entire `<html>` node.

- `node.hasChildNodes()`—Returns true/false.
- `node.hasAttributes()`—Returns true/false.
- `node.appendChild(child)`—Adds *child* as a new child node of *node*.
- `node.removeChild(child)`—Removes the indicated *child* node from the *node*.
- `node.insertBefore(child , target)`—Adds the *child* node just before the specified *target* child of this *node*.
- `node.replaceChild(child , target)`—Replaces the *target* child node with the given *child*. If *child* is a `DocumentFragment` then all its child nodes are inserted in place of *target*.

Note, if *child* is already in the DOM tree, it is first removed before becoming a new child. Section 10.14 shows how to create a new node.

10.8 DOM Tree Depth-First Traversal

Using the DOM for DHTML basically involves accessing nodes and modifying nodes on the DOM tree. The easiest way to access a target HTML element is to use

```
document.getElementById( id )
```

to obtain the node for the element directly by its *id*.

But it is also possible to reach all parts of the DOM tree by following the parent, child, and sibling relationships. A systematic visit of all parts of the DOM tree, a *traversal*, may be performed *depth-first* or *breadth-first*. In depth-first traversal, you finish visiting the subtree representing the first child before visiting the second child, etc. In breadth-first traversal, you visit all the child nodes before visiting the grandchild nodes and so on. These are well-established concepts in computer science.

Let's look at a Javascript program that performs a depth-first traversal (Ex: **DomDft**) starting from any given node on the DOM tree. The example demonstrates navigating the DOM tree to access information.

```
var result="";

function traversal(node)
{   result = "";                               // (1)
    node.normalize();                          // (2)
    dft(node);                                // (3)
    alert(result);                             // (4)
}

function dft(node)
{   var children;
    if ( node.nodeType == Node.TEXT_NODE )      // (5)
        result += node.nodeValue;
    else if ( node.nodeType == Node.ELEMENT_NODE ) // (6)
    {   openTag(node);                          // (7)
        if ( node.hasChildNodes() )             // (8)
        {   children = node.childNodes;          // (9)
            for (var i=0; i < children.length; i++) // (10)
                dft( children[i] );
            closeTag(node);                      // (11)
        }
    }
}
```

Given any **node** on the DOM tree, the **traversal** function builds the HTML source code for the node. It initializes the **result** string (line 1), normalizes the subtree rooted at **node** (line 2), calls the depth-first algorithm **dft** (line 3), and displays the result (line 4).

The **dft** function recursively visits the subtree rooted at the **node** argument. It first checks if **node** is a text node (a leaf) and, if true, adds the text to **result** (line 5). Otherwise, if **node** is an element node (representing an HTML element), it adds the HTML tag for the node to **result** by calling **openTag** (line 7), and, if there are child nodes, recursively visits them (lines 8-10) before adding the close tag (line 11). The subscript notation **children[i]** is a shorthand for **children.node(i)**.

```

function closeTag(node)
{ result += "</" + node.tagName + ">\n"; }

function openTag(node)
{ result += "<" + node.tagName;
  var at;
  if ( node.hasAttributes() )                // (12)
    tagAttributes(node.attributes);
  if ( node.hasChildNodes() )
    result += ">\n";                        // (13)
  else
    result += " />\n";                      // (14)
}

function tagAttributes(am)
{ var attr, val;
  for (var i=0; i < am.length; i++)          // (15)
  { attr = am[i]; val = attr.value;
    if ( val != undefined && val != null      // (16)
        && val != "null" && val != "" )
    { result += " " + attr.name + "=\"" +    // (17)
      val + "\"";
    }
  }
}

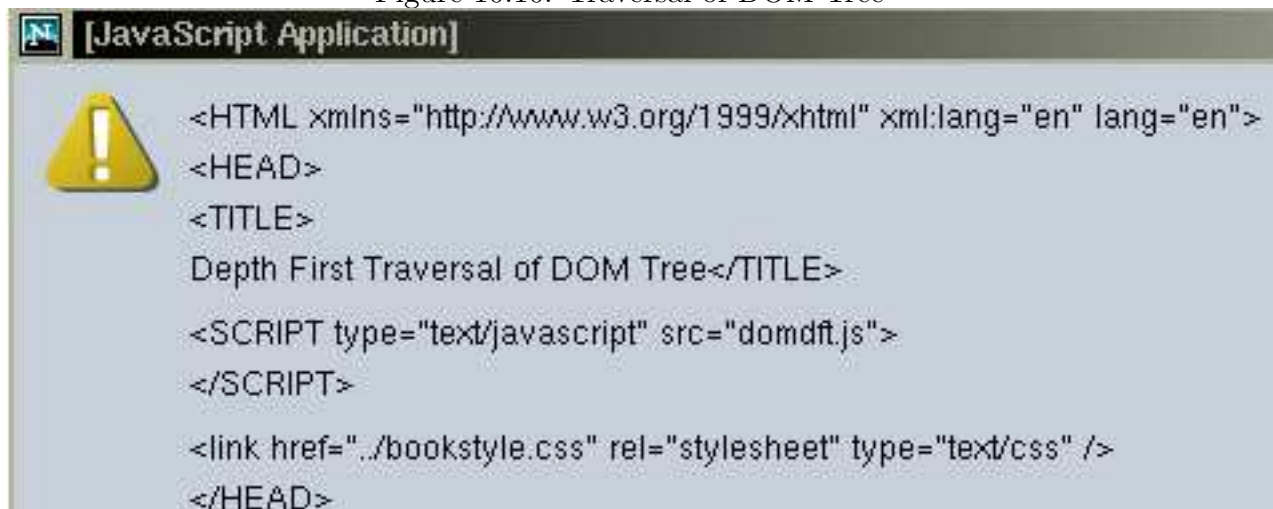
```

The `openTag` function adds any attributes for the tag (line 12) by calling `tagAttributes`. The open tag is terminated by either ">" (line 13) for non-empty elements or " />" for empty elements (line 14) conforming to the XHTML convention.

The argument `am` of `tagAttributes` is a `NamedNodeMap` of `Attr` nodes. The function goes through each attribute (line 15) and adds each defined attribute (line 16) to the `result` string (line 17). Note the use of the `name` and `value` fields of an `Attr` node.

Figure 10.10 shows the first part of the result of the depth-first traversal when called on the `document.documentElement` node corresponding to the `<html>` element of the page. The complete example can be tested on the WDP site.

Figure 10.10: Traversal of DOM Tree



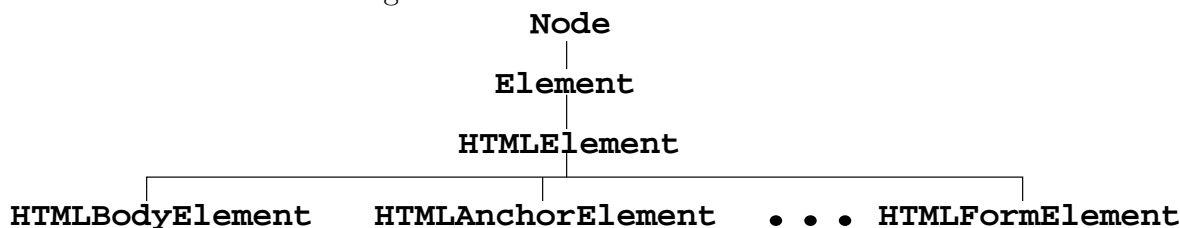
10.9 The DOM HTMLElement Interface

Derived node types (interfaces extending `Node`) add fields and methods specialized to a particular node type and may provide alternative ways to access some of the same features provided by `Node`. HTML markup elements in a page are represented by nodes extending the base `HTMLElement` which itself extends `Node`. For each element in HTML, DOM HTML provides an interface

`HTMLFullTagNameElement`

that derives from `HTMLElement` (Figure 10.11). The complete list of all the HTML element interfaces can be found in the DOM HTML specification.

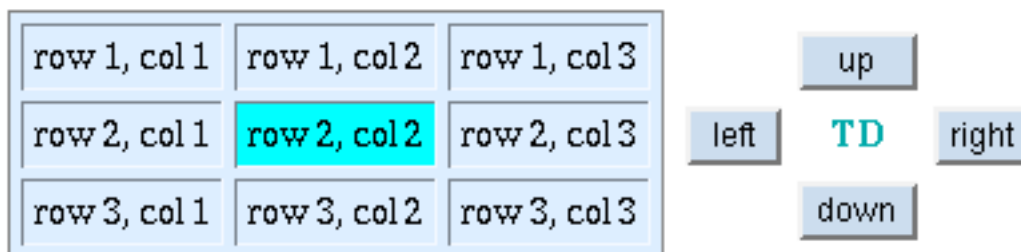
Figure 10.11: DOM HTML Interfaces



The `HTMLElement` interface is rather central for DHTML. Before we systematically discuss the fields and methods of `HTMLElement`, let's see it in action in an example (Ex: **DomNav**)

Brooks/Cole book/January 28, 2003

Figure 10.12: DOM Tree Visual Navigation



where we combine navigation and modification of the DOM tree to achieve the kind of visual effects attributable to DHTML.

We can illustrate DOM tree navigation visually by visiting a subtree representing a `<table>` element, for instance. As you traverse the subtree, the part of the table corresponding to the node being visited will be highlighted. A control panel enables you to go *up* (to the parent node), *down* (to the first child), *left* (to the previous sibling) or *right* (to the next sibling) within the table. The control panel also displays the tag name associated with the current node (Figure 10.12).

The HTML code for the table that we will be traversing is shown here in an easy-to-read form

```
<table id="tbl" border="1"
  style="background-color: #def"
  cellspacing="4" cellpadding="4" >
<tr><td>row 1, col 1</td>
  <td>row 1, col 2</td>
  <td>row 1, col 3</td></tr>
<tr><td>row 2, col 1</td>
  <td id="center">row 2, col 2</td>
  <td>row 2, col 3</td></tr>
<tr><td>row 3, col 1</td>
  <td>row 3, col 2</td>
  <td>row 3, col 3</td></tr>
</table>
```

In the actual file, we eliminate all line breaks and whitespaces between elements to avoid potential extraneous nodes on the DOM tree.

The `init()` function is executed `onload` and sets the stage for the visual navigation:

```
var currentNode, tableNode, nameNode, normal, highlight;

function init()
{
  tableNode=document.getElementById("tbl");
  tableNode.normalize();
  highlight="#0ff";
  normal=tableNode.style.backgroundColor;           // (A)
  currentNode=document.getElementById("center");    // (B)
  currentNode.style.backgroundColor = highlight;    // (C)
  nameNode=document.getElementById("tname").firstChild;
  nameNode.data=currentNode.tagName;               // (D)
}
```

The Javascript global variables used are:

- `tableNode`—the node for `<table>` that is to be traversed
- `currentNode`—the node for the current traversal position on the `tableNode` subtree
- `nameNode`—the node to display the `tagName` of `currentNode`
- `normal` and `highlight`—the background colors used to indicate visually the part of the table being visited

The `init()` function assigns initial values to these variables. The normal background color is set to the background of the table (line A). The center cell of the 3×3 table is chosen as the starting point of the traversal and `currentNode` is set (line B) and highlighted (line C). The text of `nameNode`, at the center of the control panel, is set using the `tagName` field of an `HTMLElement` (line D). The `init()` function is called `onload`:

```
<body onload="init()">
```

The control panel (Figure 10.12) for interactive traversal is another table

```
<table cellpadding="2" cellspacing="2">
<tr align="center">
  <td></td>
```

```

    <td><input type="button"
        value=" up " onclick="up()" /></td>
    <td></td></tr>
<tr align="center">
    <td><input type="button" value=" left "
        onclick="left()" /></td>
    <td id="tname" style="color: #0aa;                (E)
        font-weight: bold">tag name</td>
    <td><input type="button" value="right"
        onclick="right()" /></td></tr>
<tr align="center">
    <td></td>
    <td><input type="button" value="down"
        onclick="down()" /></td>
    <td></td></tr></table>

```

The data cell `id=tname` (line E) is used to display the tag name of the current traversal position. The four buttons each triggers a corresponding function that does the obvious. The `up()` function keeps the traversal from leaving the subtree (line F).

```

function up()
{ if ( currentNode == tableNode ) return;          // (F)
  toNode(currentNode.parentNode);
}

function down()
{ toNode(currentNode.firstChild); }

function left()
{ toNode(currentNode.previousSibling); }

function right()
{ toNode(currentNode.nextSibling); }

```

Each of these four functions calls `toNode` to visit the new node passed as the argument.

The `toNode` function does the actual work of walking from the current node to the new node given as `nd` (line G). If `nd` is `null` or a leaf node (type `TEXT_NODE`), then nothing is done (line H). If we are leaving an internal node on the subtree the highlight is removed by calling the `removeAttribute` method of the `HTMLElement` interface (line I). If we are leaving the

root `tableNode`, the original background color of the table is restored (line J). The arrival node is then highlighted and set as the current node (lines K-L). Finally, the tag name of the current node is displayed as the text content of `nameNode` (line M)

```
function toNode(nd)                                // (G)
{  if ( nd == null ||
    nd.nodeType == 3 )  // Node.TEXT_NODE          // (H)
    return false;
  if ( currentNode != tableNode )
    currentNode.style.backgroundColor="";          // (I)
  else
    currentNode.style.backgroundColor = normal;    // (J)
  nd.style.backgroundColor = highlight;            // (K)
  currentNode=nd;                                  // (L)
  nameNode.data=currentNode.tagName;               // (M)
  return true;
}
```

The example , further illustrates the DOM tree structure, use of the `style` property of HTML elements, and the `tagName` field. It also shows how DHTML can help the delivery of information, enable in-page user interactions, and enhance understanding.

You can find the complete, ready-to-run, version in the example package. You may want to experiment with it and see what it can show about the DOM tree and DHTML.

Assignment 5 suggests adding a display of the table subtree to show the current node position on the subtree as the user performs the traversal.

10.10 HTMLInputElement Fields and Methods

Every HTML element is represented on the DOM tree by a node of type `HTMLInputElement`. The `HTMLInputElement` interface extends the `Element` interface which, in turn, extends the basic `Node` interface. We list often-used fields and methods available in any node object of type `HTMLInputElement`.

- `tagName`—is a read-only field representing the HTML tag name as a string.

- **style**—is a field to a style object representing the style declarations associated with an element. For example, use `element.style.backgroundColor` to access or set the `background-color` style. Setting a style property to the empty string indicates an inherited or default style. If you change the style of an element by setting its **style** attribute instead, the new **style** attribute replaces all existing style properties on that element and, normally, that is not what you want to do. It is advisable to use the **style** field to set individual style properties you wish to change.
- **innerHTML**—is a read-write field representing the HTML source code contained inside this element as a string. By setting the **innerHTML** field of an element, you replace the content of an element. This useful field is not part of the DOM specification but supported by all major browsers.
- **getAttribute(attrName)**—returns the value of the given attribute *attrName*. The returned value is a string, an integer, or a **boolean** depending on the attribute. Specifically, a **CDATA** (character data) value is returned as a string; a **NUMBER** value is returned as an integer; an on-or-off attribute value is returned as a **boolean**. A value from an allowable list of values (e.g. `left|right|center`) is returned as a string. For an attribute that is unspecified and does not have a default value, the return value is an empty string, zero, or **false** as appropriate.
- **setAttribute(attrName, value)**—sets the given attribute to the specified string *value*.
- **removeAttribute(attrName)**—removes any specified value for *attrName* causing any default value to take effect.
- **hasAttribute(attrName)**—returns **true** if *attrName* is specified for this element, **false** otherwise.

When setting values, use lower-case strings for attribute names and most attribute values. When checking strings obtained by `tagName` or `getAttribute()`, be sure to make case-insensitive comparisons to guard against non-uniformity in case conventions. For example,

```

var nd = node1.firstChild;
var re = /table/i;
if ( re.test(nd.tagName) )
{ ... }

```

tests a `tagName` with the case-insensitive pattern `/table/i`.

HTML input control elements have these additional fields and methods

- `name` and `value`—are the name and value strings of an element to be submitted with a form.
- `focus()`—causes the input element to get *input focus* so it will receive keyboard input.
- `blur()`—causes the input element to lose input focus.
- `select()`—selects the current textual content in the input element for user editing or copying.
- `click()`—causes a click event on the element.

10.11 A Guided Form

Let's look at a practical example of DHTML where we use a combination of style, DOM, and Javascript to implement a *guided form* (Ex: **GuidedForm**). The idea is simple, we want to guide the end user visually through the form. This can be done by highlighting the input field that has keyboard focus (Figure 10.13). With a regular form, it is hard to spot which field has input focus. Thus, a guided form can be more user friendly and less confusing.

Here is the HTML code for this example:

```

<head><title>DOM Example: Guided Form</title>
<link rel="stylesheet" href="guidedform.css"
      type="text/css" title="guided form" />
<script type="text/javascript" src="guidedform.js">
</script></head>

```

Figure 10.13: A Guided Form

```

<body onload="init()" style="background-color: #def">
<form method="post" action="/cgi-bin/wb/join.cgi">
<p style="font-weight: bold; font-size: larger">
Join club.com</p>
<table width="280">
<tr><td class="fla">Last Name:</td>
    <td><input onfocus="highlight(this)"           (1)
              onblur="normal(this)"               (2)
              name="lastname" size="18" /></td></tr>
<tr><td class="fla">First Name:</td>
    <td><input onfocus="highlight(this)"
              onblur="normal(this)"
              name="firstname" size="18" /></td></tr>
<tr><td class="fla">Email:</td>
    <td><input onfocus="highlight(this)"
              onblur="normal(this)"
              name="email" size="25" /></td></tr>
<tr><td></td>
    <td><input onfocus="highlight(this)"
              onblur="normal(this)"
              type="submit" value="Join Now" /></td></tr>
</table></form></body>

```

The four input controls, last name, first name, email, and submit, call `highlight` onfocus (line 1) and `normal` onblur (line 2). The style of the form is defined in the `guidedform.css` file.

```

td.fla
{ background-color: #d2dbff;
  font-family: Arial, Helvetica, sans-serif

```

```
}

form input, select, textarea
{ background-color: #eef }           /* (3) */
```

The background color of input controls have been softened a bit from pure white (line 3).

The actual highlighting is done by these Javascript functions:

```
var base="", high;

function init() { high = "#9ff"; }    // (4)

function highlight(nd)
{ base = nd.style.backgroundColor;    // (5)
  nd.style.backgroundColor=high;      // (6)
}

function normal(nd)
{ nd.style.backgroundColor=base; }    // (7)
```

The `init()` function, called onload, defines the highlight color to use. Before highlighting an input field (line 6), its background color is saved in the global variable `base` (line 5) for later restoration. The `onblur` event handler `normal` restores the original background color (line 7).

When experimenting with this example, you can move the input focus with the mouse or forward with TAB and backward with SHIFT TAB.

10.12 Fade-in Headlines

Another useful DHTML effect provides an easy and efficient way to include page headlines that move into place as they fade in (Ex: **FadeIn**). The effect calls attention to the headline and gives the page a touch of animation.

The HTML code for such a centered headline involves

```
<body id="bd" onload="centerNow('ct', 60, 50, 40, 25)">
```

```
<p id="ct" class="headline"
  onclick="centerNow('ct', 60, 50, 40, 25)">Super ABC Company</p>
```

The style of the headline is given by a style rule such as

```
p.headline
{
  text-align: center;
  font-family: verdana, arial, helvetica;
  font-size: x-large;
  font-weight: bold;
}
```

The Javascript function `centerNow` performs the centering while fading-in animation. You specify the target color of the headline and the number of animation steps, it does the rest. The function call

```
centerNow(id, r, g, b, steps)
```

gives the *id* of the headline element (`ct` in our example), the red, green, and blue components of the target color `rgb(r%, g%, b%)`, and an integer *steps*, the number of steps for the animation.

To move the text from left to center, we use increasingly smaller right margins for the centered text. Typically, we can begin with a 60% right margin and decrease it down to 0% in the given number of steps. The code to set the right margin is

```
element.style.marginRight = setting + "%";
```

To achieve fade-in, we can repeatedly set the `color` style property

```
element.style.color = rgb(red, green, blue)
```

The animation can begin with the most-faded color and gradually change to the least-faded color (the target color) in the given number of steps. To fade a color we increase the rgb components while keeping their ratios. All colors used must keep the ratios among the rgb components as closely as possible to the original ratios in the target color. See Figure 10.14 for a sample set of headline fade-in colors.

Here is a procedure to compute the most-faded color:

Figure 10.14: Color Fade In

er ABC Company
 Super ABC Company
 Super ABC Company
 Super ABC Company
 Super ABC Company
 Super ABC Company
 Super ABC Company
 Super ABC Com

1. Let the target color be `rgb(r0%, g0%, b0%)`.
2. Let *high* be the maximum of *r0*, *g0* and *b0*.
3. Let *m* be a multiplier such that $m * high = 100$.
4. The most faded color is `rgb(m*r0%, m*g0%, m*b0%)`.

The multiplier *m* is ≥ 1 . The fade-in can then be done by multiplying the target color by a sequence of numbers from *m* to 1 in the given number of steps.

Now, let's see how this is achieved in Javascript.

```
var steps, mar, m, r, g, b;
var sty=null;

function centerNow(nd, rr, gg, bb, st)
{
  mar = 60;                                // (A)
  steps = st;                              // (B)
  r =rr;  g =gg;  b=bb;                    // (C)
  if ( sty == null )
    sty = document.getElementById(nd).style; // (D)
  margin_d = mar/steps;                    // (E)
  m = 100/Math.max(Math.max(r, g), b);    // (F)
  color_d = (m-1.0)/steps;                 // (G)
  centering();
}
```

The `centerNow` function initializes the starting right margin, the total number of animation steps, and the target color values (lines A-C). The `sty` (line D), the style of the headline element, will be used repeatedly. The percentage setting of the right margin will decrease by `margin_d` (line E) after each step. The multiplier `m` begins with the maximum value (line F) and decreases by `color_d` (line G) after each step.

With these values set, `centering()` is called to perform the actual animation.

```
function centering()
{  if (steps > 1)
    {  sty.marginRight = mar+"%";           // margin
      sty.color="rgb(" +r*m+ " %," +g*m+ " %,"
          +b*m+ " %)" ;                    // color
      mar -= margin_d;                     // decrements
      m -= color_d;
      steps--;
      setTimeout("centering()", 18);        // (H)
    }
    else // final position and color        // (I)
    {  sty.marginRight="0%";
      sty.color="rgb("+r+"%,"+g+"%,"+b+"%)" ;
    }
}
```

The `centering` function performs one step of the animation. It sets the right margin and color and decrements the quantities that will be used in the next step. The Javascript built-in function `setTimeout` (Section 9.17) schedules the call to `centering` after 18 milliseconds (line H). A smooth animation requires some 30 frames per second, making the delay between each step about 33 milliseconds. The last step of the animation (line I) makes sure we have the correct centered position and the true target color, without floating-point errors.

You can easily modify this example for similar visual effects. For example, the in-place fade-in (Figure 10.15) of a centered headline (Ex: **InPlace**) can be done with the same color technique plus changes in the letter spacing. Recall the style property `letter-spacing` controls the spacing between characters in a piece of text (Section 6.14).

With the variable `sp` set to 1 at first. The Javascript statements

Figure 10.15: In-place Fade-in



```
sty.letterSpacing = sp + "px";
sp++;
```

can be included in a fade-in function that is called for a given number of steps to fade in any target headline. The full example (Ex: **FadeIn**) can be found at the WDP site and in the example package.

10.13 Mouse Tracking

DOM also specifies an **Event** interface to provide standards for an event system, event names, event registration, and event objects.

For example a **MouseEvent** object has the **clientX** and **clientY** fields giving, respectively, the x and y coordinates of the mouse event position in the document display area. Using these coordinates associated with the **mousemove** event, we can *drag* an element by moving the mouse.

The following HTML code displays an image, a crystal ball, that when clicked will follow the mouse until the mouse is clicked again (Ex: **DragDrop**).

```
<head><title>Drag and Drop</title>
<script type="text/javascript" src="dragdrop.js"></script>
</head> <body onload="init()">
<div id="ball" onclick="drag()"                                (1)
    style="position: absolute;
        top: 20px; left: 20px; z-index: 1">

</div></body>
```

The technique is straight forward. A mouse click calls the Javascript function `drag()` (line 1) that sets up the `trackMouse` event handler for the `mousemove` event (line 3). Mouse tracking changes the absolute position of the crystal ball. The `left` and `top` style properties are set to the event coordinates plus any scrolling that may have taken place for the browser window (lines 5-6).

```
// file: dragdrop.js
var ball, ballstyle;

function init()
{  ball = document.getElementById('ball');
   ballstyle = ball.style;
}

function drag()
{  if ( document.onmousemove )
    document.onmousemove = null;           // (2)
    else
    document.onmousemove = trackMouse;      // (3)
}

function trackMouse(e)                     // (4)
{  var x = e.clientX + window.scrollX;     // (5)
   var y = e.clientY + window.scrollY;     // (6)
   ballstyle.left = x + "px"; // left style property
   ballstyle.top = y + "px";  // top style property
}
```

A second mouse click calls `drag()` and cancels the mouse tracking (line 2).

10.14 The DOM HTMLDocument Interface

Browsers display Web pages in windows. Each window has a unique `document` object that represents the entire Web page displayed in the window. The `document` object contains all other elements in the page.

The `document` object, implementing the DOM `HTMLDocument` interface which inherits from the `Document` interface, offers fields and methods useful for page-wide operations.

HTMLDocument Fields

A select set of fields available from the `document` object is listed here.

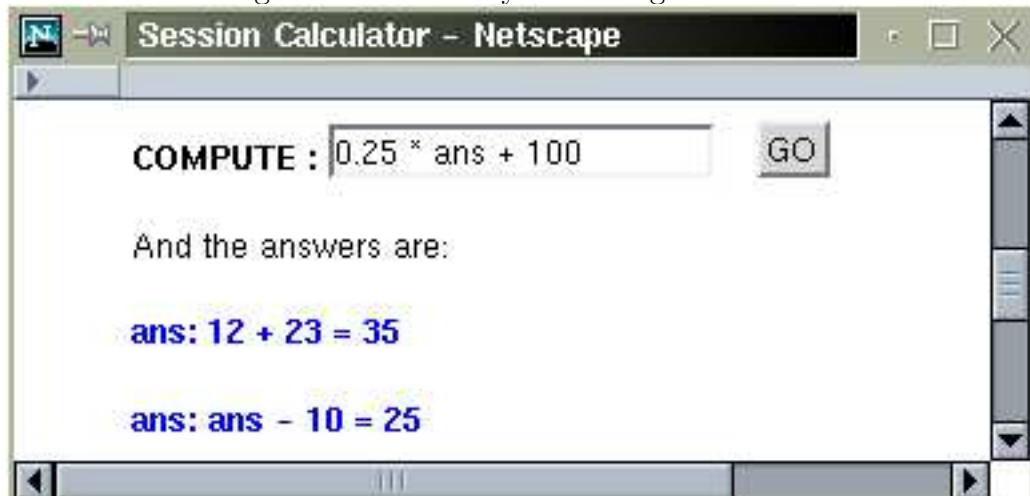
- `documentElement`—the `<html>` element of the page.
- `body`—the `<body>` element of the page.
- `URL`—a read-only string for the complete URL of the page.
- `title`—the title string specified by `<title>`.
- `referrer`—the read-only URL of the page leading to this page (empty string if no referrer).
- `domain`—the read-only domain name of the Web server that supplied the page.
- `cookie`—a SEMICOLON-separated string of *name=value* pairs (the cookies) associated with
- `anchors`, `applets`, `forms`, `images`, `links`—read-only lists of different elements in the page: `<a>` elements as named anchors, `<applet>` elements, `<form>` elements, `` elements, and `<a>` and `<area>` elements as `href` links, respectively. Such a list has a `length` field, an `item(n)` method, and a `namedItem(name)` method which returns an element with *name* as `id` or, failing that, as `name`.

HTMLDocument Methods

Frequently used methods of the `document` object include:

- `createElement(tagName)`—returns a newly created element object for the `<tagName>` element. By setting attributes and adding child nodes to this element, you can build a DOM structure for any desired HTML element.
- `createTextNode(textString)`— returns a node of type `TEXT_NODE` containing the given `textString`.

Figure 10.16: History Recording Calculator



- `getElementById(id)`— returns the unique HTML element with the given *id* string. We have seen this method used often.
- `getElementsByTagName(tag)`—returns a list of all elements with the given *tag* name in the document.

10.15 Generating New Content

Applying the features discussed in the previous section, let's do more with DHTML by adding new content to a displayed HTML page. The content is computed by Javascript, built into element nodes, and placed on the DOM tree.

A Session-Recording Calculator

To get started, we can take the interactive calculator example (Ex: **DomCalc**) shown in Figure 10.6 and make it more useful by recording the current answer and displaying a history of computation steps. The answer from the previous step can be used in the next step (Figure 10.16).

The HTML code for the session calculator (Ex: **DomSession**) is revised slightly from

element around it (lines F-G), and appends the element as a new (last) child of the `session` `<div>` (line H). Finally, the input field is cleared (line I), ready for the next step. Users may use `ans` in the next computation step to perform a session of steps (Figure 10.16). Further, users may store values for use in subsequent steps by creating their own variables with input strings such as:

```
taxRate = 0.08
total = subtotal + subtotal * taxRate
```

10.16 A Smart Form

As another example of dynamically adding and removing page content, let's add some smarts to the guided form discussed in Section 10.11 (Figure 10.13).

Figure 10.17: Smart Form I



Full Name: Paul Wang

Country: Canada

Telephone: ###-###-####

Join Now

A website in North America may collect customer address and telephone information without asking for an *international telephone country code*. But, if the customer selects a country outside of North America, it may be a good idea to require this information as well. In many situations, the information to collect on a form can depend on data already entered on the form. It would be nice to have the form dynamically adjust itself as the user fills out the form. This can be done with DHTML.

As an example, let's design a smart form (Ex: **SmartForm**) that examines the country setting in the address part of the form and adds/removes (Figures 10.17 and 10.18) an input field for the international telephone code.

Our strategy is straight-forward:

1. When the country name is selected, the **onchange** event triggers a call to check the country name.
2. Any country outside North America causes an input field to be added to obtain the telephone country code.
3. If the country is inside North America, then any telephone country code input field is removed.

Figure 10.18: Smart Form II

The HTML code is as follows:

```
<head><title>DOM Example: Smart Form</title>
<link rel="stylesheet" href="guidedform.css"
      type="text/css" title="dynamic guided form" />
<script type="text/javascript" src="smartdform.js">
</script>
</head>
<body onload="init()" style="background-color: #def">
<form method="post" action="http://cgi-bin/join.pl">
<p style="font-weight: bold; font-size: larger">
Join club.com</p>
```

Brooks/Cole book/January 28, 2003

```

<table><tbody id="tb">                                     (A)
<tr><td class="fla">Full Name:</td>
  <td><input onfocus="highlight(this)"
    onblur="normal(this)"
    name="fullname" size="20" /></td></tr>
<tr><td class="fla">Country:</td>
  <td><select id="country" name="country"
    size="1"
    onfocus="highlight(this);"
    onchange="countryCode(this);"                (B)
    onblur="normal(this);" >
    <option value="US">USA</option>
    <option value="CA">Canada</option>
    <option value="MX">Mexico</option>
    <option value="CN">China</option>
    <option value="RU">Russian Federation
    </option>
  </select></td></tr>
<tr><td class="fla" >Telephone:</td>
  <td><input onfocus="highlight(this)"
    onblur="normal(this)"
    name="phone" size="20" /><span id="pinst">        (C)
    ###-###-####</span></td></tr>
<tr id="bt"><td></td>                                     (D)
  <td><input onfocus="highlight(this)"
    onblur="normal(this)"
    type="submit" value="Join Now" /></td>
</tr></tbody></table></form></body>

```

The `onchange` event of `<select>` triggers the function `countryCode` (line B) which can add/remove a form entry for the telephone country code. The new form entry will be a new table row element, a child `<tr>` of `<tbody>` (line A) inserted just before the row (line D) for the submit button.

The `init()` function, executed onload, sets the telephone instruction node (`inode`) to the `pinst` span (lines C and 1). The text child of `inode` (line 2) can be replaced later by a generic instruction (`oph`) for other countries (line 3).

```
var oph, iph, inode;
```

```

function init()
{  inode = document.getElementById("pinst");    // (1)
  iph = inode.firstChild;                      // (2)
  oph = document.createTextNode(
    " AreaCode-Phone Number");                // (3)
}

```

Two additional global variables (line 4) are used: `crow` (the table row to be created) and `cc` (the `<input>` element for the telephone country code). The `isLocal` function checks to see if a country is local to North America (line 5).

```

var crow = null, cc=null;                      // (4)

function isLocal(ct)                          // (5)
{  return (ct == "US" || ct == "CA"
        || ct == "MX" );
}

function countryCode(country)
{  var d1, d2, t1, t2, button;
  var tbody = document.getElementById("tb");
  if ( isLocal(country.value) )                // (6)
  {  if ( crow != null )
    {  tbody.removeChild(crow);               // (7)
       inode.replaceChild(iph, oph);
       cc = crow = null;                      // (8)
    }
    return;
  }
  // country outside North America
  if ( crow != null )                          // (9)
  {  cc.value = ""; return; }                 // (10)

  crow = document.createElement("tr");        // (11)
  crow.appendChild(makeLabel());              // (12)
  crow.appendChild(makeCC());                 // (13)

  button = document.getElementById("bt");     // (14)
  tbody.insertBefore(crow, button);          // (15)
  inode.replaceChild(oph, iph);               // (16)
}

```

A call to `countryCode` is triggered by the `onchange` event on the `<select>` element for the country part of an address. If the given `country` is in North America, it removes any telephone country code entry from the form, restores the phone instructions, resets the global variables, and then returns (line 6-8).

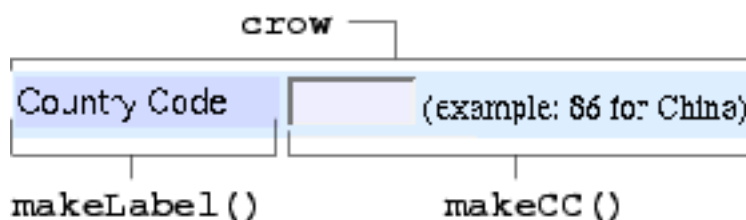
The function continues to process `country` which is outside of North America. If the telephone country code entry is already displayed (line 9), it simply makes sure any previously entered code is removed and returns (line 10). Otherwise, a new `<tr>` element is created (line 11), filled with two table cells (lines 12-13), and inserted into the table body just before the submit button (lines 14-15). The generic phone instruction is also put in place (line 16)

Each of the following functions makes a `<td>` element for the table row needed (Figure 10.19). The `input` element is made to match the style and dynamic behavior of other input controls in the form (lines 17-18). A text label is created by `makeLabel()` and the actual input element for the telephone country code is created by `makeCC()` which also sets the global variable `cc`.

```
function makeLabel()
{
  var t, d;
  d = document.createElement("td");
  d.setAttribute("class", "fla");
  t = document.createTextNode("Country Code:");
  d.appendChild(t);
  return d;
}

function makeCC()
{
  var t, d;
  d = document.createElement("td");
  cc = document.createElement("input");           // (17)
  cc.setAttribute("onfocus", "highlight(this)");
  cc.setAttribute("onblur", "normal(this)");
  cc.setAttribute("name", "cc");
  cc.setAttribute("id", "cc");
  cc.setAttribute("size", "7");                   // (18)
  d.appendChild(cc);
  t = document.createTextNode("(example: 86 for China)");
  d.appendChild(t);
}
```

Figure 10.19: Creating A Form Entry



```

    return d;
}

```

Note these element creation functions use `setAttribute` to set up many attributes so the newly created form entry fits in with the style and dynamic behavior on this smart form.

Experiment with Ex: **SmartForm** and see for yourself.

One small problem for this form shows up when you click the **back** button after submitting the form, perhaps because of an error in filling the form. The form contains the data you entered, but the telephone code entry disappears. It is possible to add code to the `init()` function called onload to fix this problem and the solution is left as an exercise (Assignment 8).

10.17 Reordering Rows in Tables

Applying DHTML, we can make tables more usable by allowing the end user to reorder rows based on the contents of cells in any given table column. Thus, tables representing invoices, shopping carts, airfares, addresses, student grades, and so on, can be sorted at will by the end user. The user may want to list the largest amount first, the least expensive item first, or names alphabetically. With DHTML, the user can do this by clicking the mouse and the sorting will be performed by client-side Javascript. Not going back to the server, the redisplay is instantaneous and very dynamic.

For example, the shopping cart in Figure 10.20 is in increasing **Amount**. The same shopping cart is shown in Figure 10.21 in increasing unit **Price**.

Figure 10.20: Shopping Cart Sorted by Amount

Your Cart

Item	Code	Price	Quantity	Amount
Shovel	G01	14.99	2	29.98
Power Saw	P12	34.99	1	34.99
Hand Shovel	T01	4.99	10	49.90
Hand Saw	H43	24.99	5	124.95
Subtotal:				239.82

In this example (Ex: **TableSort**), clicking (double-clicking) on a table header cell sorts that column in increasing (decreasing) order.

Sortable Table Organization

The HTML code for the sortable table is organized as follows.

- The first table row is placed within a `<thead>` element and contains table header (`<th>`) cells connected to event `onclick` and `ondblclick` handlers. These cells have a button look to visually indicate their active nature. Here is a typical `<th>` cell:

```
<th class="button"
  onclick="sortTable(2, 'num', '1');"
  ondblclick="sortTable(2, 'num', '-1');">Price</th>
```

The arguments to `sortTable` are column position (a zero-based index), numerical or alphabetical ordering (`num` or `str`), and increasing or decreasing order (`1` or `-1`). Here is the header for the `Item` column

```
<th class="button"
  onclick="sortTable(0, 'str', '1');"
  ondblclick="sortTable(0, 'str', '-1');">Item</th>
```

- The sortable table rows are organized in a `<tbody>` group with a given `id`, `tb` in our example. Each sortable column must contain all numbers or all text strings. Here is a typical row

```
<tr id="cc" valign="middle" align="right">
  <th>Hand Shovel</th>
  <td align="center">T01</td>
  <td>4.99</td> <td>10</td> <td>49.90</td></tr>
```

The table layout and the button look for clickable table header cells are created with CSS rules (Section 6.12):

```
table.sort tr { background-color:#f0f0f0; }

table.sort th.button
{   background-color: #fc0; border-width: 3px;
    border: outset; border-color: #fc0
}
```

Table Sorting

Now let's look at the Javascript code for table sorting.

As stated in the previous subsection, `onclick` and `ondblclick` events on an active table header trigger calls to the `sortTable` function with appropriate arguments: column position (`c`), numerical or alphabetical ordering (`n`), and increasing or decreasing direction(`d`).

```
var col=null, numerical=false, direction=1;

function sortTable(c, n, d)
{   if ( col==c && Number(d)==direction ) return;           // (a)
    col=c;                                                    // (b)
    direction = Number(d);
```

Brooks/Cole book/January 28, 2003

Figure 10.21: Shopping Cart Sorted by Price

Your Cart

Item	Code	Price	Quantity	Amount
Hand Shovel	T01	4.99	10	49.90
Shovel	G01	14.99	2	29.98
Hand Saw	H43	24.99	5	124.95
Power Saw	P12	34.99	1	34.99
Subtotal:				239.82

```

numerical = (n == "num");           // (c)
var tbody = document.getElementById("tb"); // (d)
var r = tbody.childNodes;          // (e)
n = r.length;
var arr = new Array(n);
for ( i=0; i < n; i++ ) arr[i]=r.item(i); // (f)
quicksort(arr, 0, n-1);              // (g)
for ( i=0; i < n; i++ ) tbody.appendChild(arr[i]); // (h)
}

```

If `c` is the same as the recorded column position `col` and `d` the same as the recorded sorting direction (line `a`), the sorting has already been done and the `sortTable` returns immediately. To prepare for sorting, the arguments are stored in the global variables (lines `b-c`). The child nodes of `<tbody>` are copied into a new array `arr` (lines `d-f`). The copying is needed because `tbody.childNodes` is a read-only list. The notation (line `f`)

```
r.item(i)
```

gets you the `i`-th item on the list of child nodes. It is possible that the notation `r[i]` will also work.

The call to `quicksort` (line `g`) sorts the array `arr` with the quicksort algorithm, one of the most efficient sorting algorithms known. The elements on the sorted `arr` are then appended in sequence as children of `<tbody>` (line `g`).

Inserting existing nodes from the DOM tree into the DOM tree is very different from inserting newly created nodes (Section 10.16). An existing node is first removed from the DOM tree automatically before it is inserted. The removal is necessary to protect the structure integrity of the DOM tree. This is why no explicit removal of child nodes from `<tbody>` is needed before appending the nodes from the sorted array `arr`.

If you accept the `quicksort` function as a magical black box that does the sorting, then we have completed the description DHTML table sorting.

For those interested, the inner workings of `quicksort` are presented next.

Quicksort

The basic idea of the quicksort algorithm is simple. First pick any element of the array to be sorted as the *partition element* `pe`. By exchanging the elements, the array can be arranged so all elements to the right of `pe` are greater than or equal to `pe`, and all elements to the left of `pe` are less than or equal to `pe`. Now the same method is applied to sort each of the smaller arrays on either side of `pe`. The recursion is terminated when the length of the array becomes less than 2.

```
function quicksort(arr, l, h)
{   if ( l >= h || l < 0 || h < 0 ) return;      // (1)
    if ( h - l == 1 )                          // (2)
    {   if (compare(arr[l], arr[h]) > 0)        // (3)
        {   swap(arr, l, h) }                  // (4)
        return;
    }
    var k = partition(arr, l, h);               // (5)
    quicksort(arr, l, k-1);                     // (6)
    quicksort(arr, k+1, h);                     // (7)
}
```

The `quicksort` function is called with the array to be sorted, the low index `l` and the high index `h`. It sorts all elements between `l` and `h` inclusive. If the sorting range is empty (line 1), `quicksort` returns immediately. If the range has just two elements (line 2), they are compared (line 3) and switched (line 4) if necessary; and `quicksort` returns. For a wider range, `partition` is called to obtain a partition element and the left and right parts of the array. Each of these two parts is sorted by calling `quicksort` (lines 6-7).

The call `compare(a, b)` compares the arguments and returns a positive, zero, or negative number depending for $a > b$, a equals b , or $a < b$. The signs are reversed for sorting in decreasing order.

```
function compare(r1, r2)
{   ke1 = key(r1, col);                // (8)
    ke2 = key(r2, col);
    if ( numerical )                   // (9)
    {   ke1 = Number(ke1);
        ke2 = Number(ke2);
        return direction * (ke1 - ke2);
    }
    return (direction * strCompare(ke1, ke2)); // (10)
}
```

For sorting HTML tables, `compare` is called with DOM nodes `r1` and `r2` representing two different table rows. The function `key` obtains the string content in the designated table cell (line 8) and compares them either numerically as numbers (line 9) or as alphabetically as text string (line 10).

The function `key` extracts the textual content (line 12) of the table cell from the given row `r` at the column position `c`.

```
function key(r, c)
{   var cell = r.firstChild;
    while ( c > 0 )
    {   cell = cell.nextSibling;
        c--;
    }
    return cell.firstChild.nodeValue; // (12)
}
```

The `strCompare` function compares two text strings `a` and `b` by comparing corresponding characters.

```
function strCompare(a, b)
{
    var m = a.length;
    var n = b.length;
    var i = 0;
    if ( m==0 && n==0 ) return 0;
    if ( m==0 ) return -1;
    if ( n==0 ) return 1;
    for ( i=0; i < m && i < n; i++ )
    {
        if ( a.charAt(i) < b.charAt(i) ) return -1;
        if ( a.charAt(i) > b.charAt(i) ) return 1;
    }
    return (m - n);
}
```

And swapping two elements on the array is simple.

```
function swap(arr, i, j)
{
    var tmp = arr[i];
    arr[i]=arr[j];
    arr[j]=tmp;
}
```

Now we can turn our attention to `partition`, the work horse in the `quicksort` algorithm. The function is called with an array `arr`, and a sorting range, defined by the low index `l` and the high index `h`, which has at least 3 elements. The function picks the middle element as the `pe` (line 13), and partitions the given range into two parts separated by the `pe`. All elements to the left of `pe` are less then or equal to `pe` and all elements to the right of `pe` are greater than or equal to `pe`. The index of the `pe` is returned (line 18).

```
function partition(arr, l, h)    // h > l+1
{
    var i=l, j=h;
    swap(arr, ((i+j)+(i+j)%2)/2, h);           // (13)
    var pe = arr[h];
    while (i < j)
    {
        while (i < j && compare(arr[i], pe) < 1) // (14)
            { i++; } // from left side
    }
```

```

        while (i < j && compare(arr[j], pe) > -1) // (15)
        { j--; } // from right side
        if (i < j) { swap(arr, i++, j); } // (16)
    }
    if (i != h) swap(arr, i, h); // (17)
    return i; // (18)
}

```

Searching from the left (line 14) and right (line 15) end of the range, it looks for a pair of out-of-order pair of elements and swaps them (line 16). When done, it moves the `pe` back into position (line 17) and returns.

The complete `quicksort` and the table sorting example can be found in the example package.

10.18 A TicTacToe Game

With DHTML many kinds of interactive games can be implemented. Let's look at TicTacToe as an example (Ex: **TicTacToe**). A CSS controlled `<table>` can server as the playing board. Moves are made by clicking on the game board squares. Two files `x.gif` and `o.gif` provide the graphical images for the game tokens (Figure 10.22).

The HTML code for the game board is a 3 by 3 `<table>`:

```

<table class="tic" cellspacing="0" border="0">
<tr><td id="tl" onclick="play('tl')"           (a)
    width="37" height="44">&nbsp;</td>
    <td id="tc" onclick="play('tc')"           (b)
    width="37" height="44">&nbsp;</td>
    <td id="tr" onclick="play('tr')"           (c)
    width="37" height="44">&nbsp;</td>
</tr>

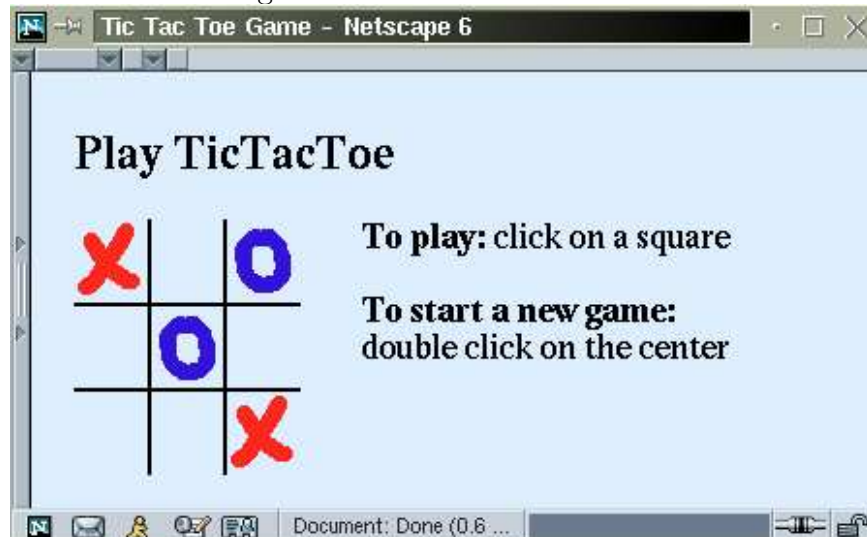
...

</table>

```

The first row is shown here. The other rows are entirely similar. Each table cell (`<td>`)

Figure 10.22: TicTacToe Game



has an `id` and a `class` attribute set to a string such as `t1` (top-left line a) and `tr` (top-right line c). We thus simply identify the nine game squares. Each `<td>` has a `width` and `height` setting to accommodate the game token image that, when played, will replace the non-breaking space ` ` place holder as the content of `<td>`. Onclick, each `<td>` calls `play` with its own `id`.

The game board is drawn with CSS `border` settings. For example, the style for the top left square is

```
td#t1      // id selector
{ border-right: thin #000 solid;
  border-bottom: thin #000 solid;
}
```

And the style for the center square is

```
td#cc
{ border-top: thin #000 solid;
  border-left: thin #000 solid;
  border-right: thin #000 solid;
  border-bottom: thin #000 solid;
}
```

These table cell styles when combined with the `cellspacing` and `border` attributes for `<table>`, draws the game board.

In the Javascript code, nine variables (line 1) are used to indicate whether a game square is open (zero) or taken (nonzero). The function `play` plays a token on the given (`id`) position. It returns immediately if the square is not open (line 2). Otherwise, it proceeds to place a token in the target square. It sets `cell` to the target `<td>` element (line 3), marks the position as taken by player one or player two (line 4), obtains an element object representing the game token (line 5), saves a copy of the content of `<td>` in the global variable `sp` (line 6), and replaces the content of `cell` with the token (lines 7-8).

```
var tl=0, tc=0, tr=0, cl=0, cc=0,
    cr=0, bl=0, bc=0, br=0;           // (1)
var which=false, sp=null;

function play(id)
{   if ( eval(id) > 0 ) return;        // (2)
    var cell = document.getElementById(id); // (3)
    eval(id + (which ? "= 1;" : "=2;")); // (4)
    tnode = token();                  // (5)
    if ( sp == null )
        sp = cell.firstChild.cloneNode(true); // (6)
    cell.removeChild(cell.firstChild); // (7)
    cell.appendChild(tnode);          // (8)
}
```

The saved `sp` is used when restoring the game board for another game. Note, the call `cloneNode(true)` performs a *deep copy* of a node, copying all nodes in the subtree rooted at the node. If the argument `false` is given, only the children of `node` will be copied.

The `token` function creates a new `` element for `x.gif` or `o.gif` depending on the setting of the Boolean variable `which` (line 9) whose value alternates every time `token` is called (line 10).

```
function token()
{   var t = document.createElement("img");
    if ( which )                       // (9)
        t.setAttribute("src", "o.gif");
```

```

    else
        t.setAttribute("src", "x.gif");
    which = ! which;           // (10)
    t.setAttribute("width", "35");
    t.setAttribute("height", "40");
    t.style.display = "block"; // (11)
    return t;
}

```

The token image displays as a block element (line 11) so it fits exactly on the board. The center square (`id=cc`), calls `newgame()` on double click which restores the game board (line 12) and resets the game variables (line 13) for another game.

```

function newgame()
{
    blank(tl, "tl"); blank(tc, "tc");           // (12)
    blank(tr, "tr"); blank(cl, "cl");
    blank(cc, "cc"); blank(cr, "cr");
    blank(bl, "bl"); blank(bc, "bc");
    blank(br, "br");
    tl=tc=tr=cl=cc=cr=bl=bc=br=0;             // (13)
}

function blank(n, id)
{
    if ( n == 0 ) return; // no token
    var cell = document.getElementById(id);
    cell.removeChild(cell.firstChild);
    if ( sp != null )
        cell.appendChild(sp.cloneNode(true));
}

```

The function `blank` replaces a board position with a token by a copy of the saved blank node `sp`. The function `blank` can be simplified to

```

function blank(n, id)
{
    if ( n == 0 ) return; // no token
    var cell = document.getElementById(id);
    cell.innerHTML = "&nbsp;";
}

```

for browsers that support the `innerHTML` feature. Under NN and IE all HTML elements have the `innerHTML` field that gives you the HTML code contained inside the element. You can also set this field to modify the content of any element. It is very convenient for programming.

This simple implementation is functional enough to be used by two players. Chapter-end exercises suggest improvements to this program. `isecionWindows` and `Frames`

Before the introduction of DOM, browsers such as NN and IE already had their own objects for windows, frames, and various HTML elements. DOM makes the document model more complete, systematic, and uniform across all browsers. The DOM addresses only the object structure of the document. The window object that represents the on-screen window containing the document is not part of the DOM specification. Many important features of the `window` object do work consistently across major browsers (Section 9.12). In particular `window.document` gives you the root of the DOM tree.

With the introduction of DOM, browser vendors are implementing the window object in the same DOM spirit leading to a more functional, and better defined interface to the `window` object. Since version 6, NN is leading the way in this regard. Materials in this section show how the `window` object works with frames and the DOM tree in NN 6 and later versions.

The window Object

Useful fields of `window` include

- `window.document`—a reference to the root of the DOM tree conforming to the DOM Document interface.
- `window.frames`—an array of frames in this window.
- `window.innerHeight`, `window.innerWidth`—the height and width in pixels for content display in the window.
- `window.navigator`—a reference to the `navigator` object (Section 9.11).
- `window.parent`—the parent window containing this window or `null`.

- `window.top`—the top-level window.
- `window.screen`—an object representing the computer display screen.

Many window methods have been described in Section 9.12. We list a few more here.

- `window.dump(str)`—outputs *str* to the Javascript console.
- `window.print()`—prints `window.document`

Vertical Page Positioning

Sometimes, a page layout calls for the positioning of an HTML element at a certain vertical position in the display window. For example a site entry may be a graphics image or a Flash animation that is vertically centered in the window or starts 1/3 of the way down. Vertical centering is not easy with HTML alone. But can be done rather simply with DHTML.

Here is a simple example (Ex: **VCenter**).

```
<body onload="vcenter()" style="margin: 0">
<table width="100%" cellpadding="0" cellspacing="0">           (1)
<tr><td></td></tr>      (2)
<tr align="left">
<td style="width:100%; height:300px"><a href="main.html">
  </a></td>
</tr></table></body>
```

The entry graphic `entry.gif` is 300 pixels high. It needs to be centered vertically in the window. We put the graphic in the second row of a table that covers the whole window (line 1). The first row is a padding provided by a transparent image (line 2) whose height is set dynamically by the Javascript function `vcenter()`:

```
function vcenter()
{  var ht = window.innerHeight-300;           // (3)
   ht = (ht - ht%2)/2;                         // (4)
   var cell = document.getElementById("padding"); // (5)
```

```

        cell.setAttribute("height", ht);                // (6)
    }

    window.onresize=vcenter;                            // (7)

```

To center vertically, `vcenter` computes the difference of the window inner height and the height of the image to be centered (line 3). Dividing that by 2 gives the height of the desired padding which is set as the height attribute of the padding image (lines 4-6). The function is called on load and also on window resizing (line 3) so the element stays centered. This idea is easily generalized to perform other dynamically computed vertical positioning.

The WDP website provides the complete working version of this example.

10.19 A Code Experimenter

For people learning or using HTML, style sheets, and Javascript, it is often beneficial to experiment with code fragments to see their effects. We can build a page with two frames, side-by-side (Ex: **TryCode**). In one frame, the user can enter and edit code in a textarea. With DHTML, the other frame can show the effect of the code (Figure 10.23).

The `frameset` page has the following code.

```

<head>
<title>Code Experimenter</title></head>
<frameset cols="40%,60%">
    <frame frameborder="1" id="codeframe" src="code.html"
        scrolling="auto" />
    <frame id="resultframe" src="result.html" />
</frameset>
</html>

```

The `result.html` page is very simple.

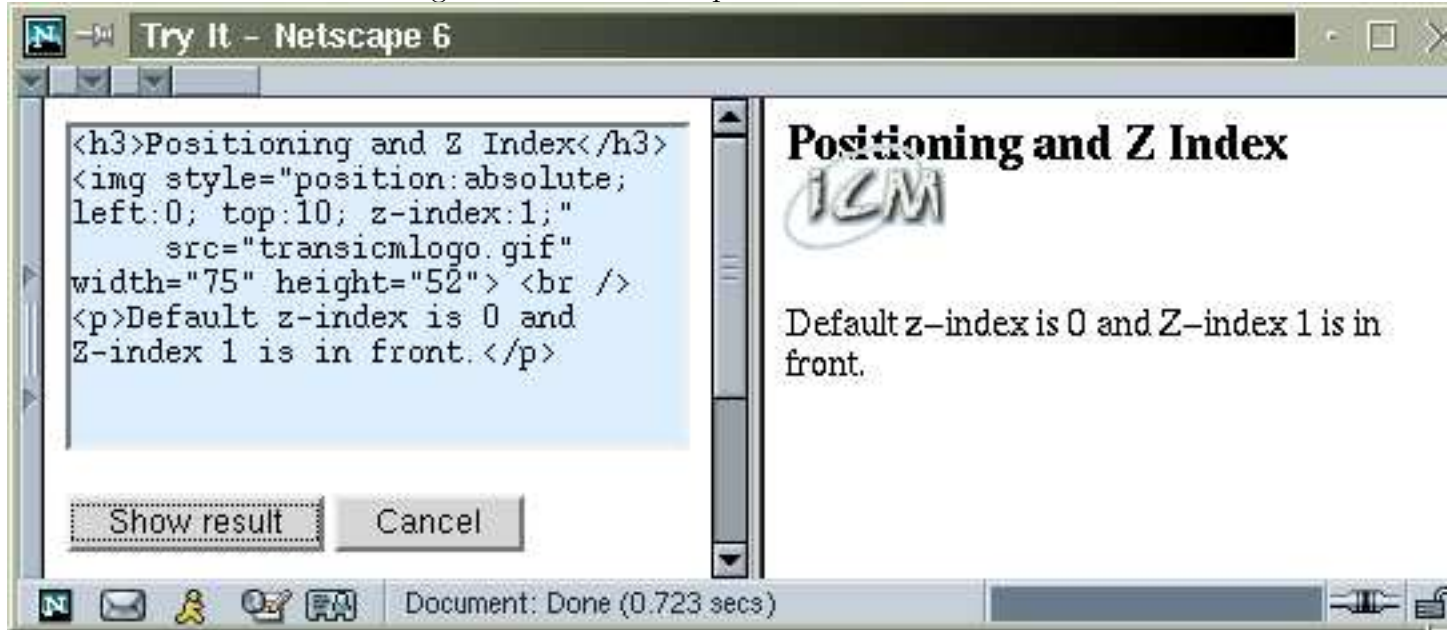
```

<head><title>Results</title></head>
<body><p>The result is shown here</p></body>
</html>

```

And the `code.html` page contains the `textarea` for entering and editing experimental code.

Figure 10.23: Code Experimenter



```
<textarea style="width:100%; background-color: #def"
          id="code" cols="65" rows="16">
```

Put your HTML code fragment here. (1)

Anything that the HTML body
element may contain is fine. (2)

```
</textarea>
```

```
<p><input name="result" type="button"
      value="Show result" onclick="show()" />
  <input type="button" value="Cancel"
  onclick="goBack()" /></p>
```

```
<p>You can enter and edit code in this window,
and click on "Show result" to see the result in
the window on the right. Click "Cancel" to go
back to the hands-on page.</p>
```

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

The HTML source contained in `textarea` (lines 1-2) can be anything here. The user will enter the code interactively for experimentation. The Javascript code in `code.html` supports the desired effects. The `Show result` button calls `show()`. It obtains the `document` object in the window for the frameset (line A). Note `window` is the window for the `codeframe` and from this frame `window.top` (or `window.parent` is the window for the frameset. From

the `resultframe` element (line B), we obtain its `document` object (line C) and the `<body>` element (line D) in that document to show the code. We applied the `HTMLDocument` method `getElementsByTagName` (Figure 10.14) to get the `body` element.

The HTML source in the `textarea` is assigned as the content `innerHTML` of the `<body>` element (lines E-F).

```
<head>
<script type="text/javascript">
function show()
{   var b = window.top.document;           // (A)
    var f = b.getElementById("resultframe"); // (B)
    var d = f.contentDocument;             // (C)
    var bb = d.getElementsByTagName("body").item(0); // (D)
    var c= document.getElementById("code"); // (E)
    bb.innerHTML=c.value;                  // (F)
}

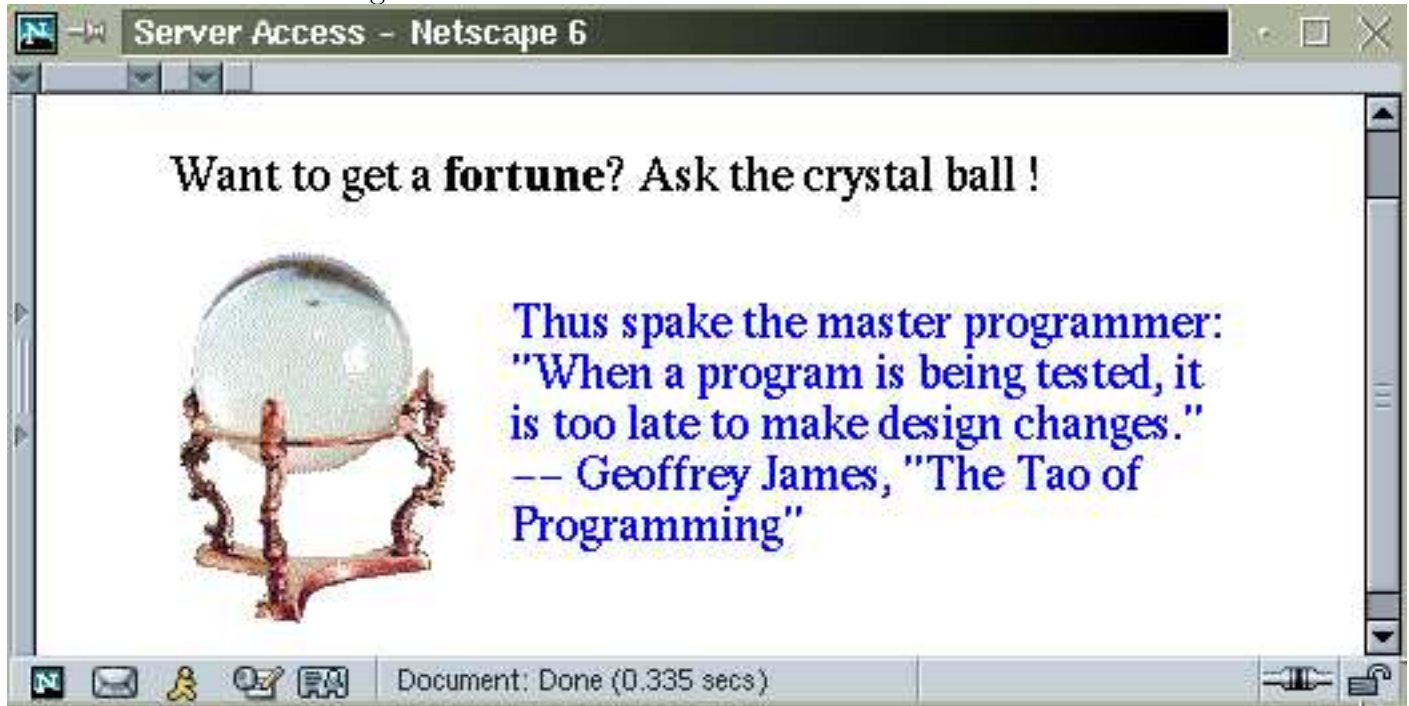
function goBack() { window.top.back(); }
</script>
```

This example also shows the usage of frame objects and how they can interact under DOM.

10.20 DHTML Access to Web Services

DHTML can do many things to make Web pages more responsive and more functional. But it is, up to this point, restricted to operating on data already in the page and input by the user from the keyboard. DHTML can be much more powerful and useful if it can manipulate data from other Web pages and perhaps even obtain data from server-side programs. The data thus obtained can be incorporated at appropriate places in a Web page. For example, a stock price can be obtained and inserted in an article on stock trading; a travel page can display weather forecast information of the departure and arrival cities for the travel dates; maps and driving directions can be included for stores and businesses, etc.

Figure 10.24: Fortune Cookie Web Service



Thus, *Web services* provide well-defined data and DHTML can fetch such data dynamically and insert them at designated places on the DOM tree. A Web service does not have to return a complete HTML page. It may return just an HTML fragment suitable for inclusion in another Web page. In time such usage will be common-place and standardized.

To show how this can work with current DHTML techniques, let's look at a *fortune cookie* program (Figure 10.24). This program Ex: **Fortune** works with two frames as follows.

- When the end user clicks on the picture of a crystal ball, a request is made to a Web service that returns a **fortune cookie** message.
- The fortune cookie page is loaded in a hidden frame and contains Javascript code which will deposit the fortune cookie HTML fragment into the viewing frame.

The frameset code hides the first frame (line 1) and shows only the second frame (line 2).

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

```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head><title>Web Service Access</title></head>
<frameset rows="0,*">
    <frame id="hide" name="hide" src="empty.html" />           (1)
    <frame id="show" name="show" src="fortune.html" />         (2)
</frameset>
</html>

```

The `empty.html` file is very simple and can be anything similar to

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

The `fortune.html` page displays the crystal ball and ties it to the access of the fortune cookie service (line 3).

```
<head>
<title>Web Service Demo: Fortune Cookie</title>
<script type="text/javascript" src="service.js"></script>
</head>

<body style="font-size: Large; margin: 50px 10%">
<p >Want to get a <b>fortune</b>?
    Ask the crystal ball !</p>
<p>
</p>
<p id="ans" class="ans">
000<!--*@!+_( $&#0000000<!--*@!+_( $      // random letters
&#00000000000<!--*@!+_( $&#</p>
</body>
```

The `comp` function is in `service.js`. It obtains the frameset window (line 4), then the window object for the hidden frame (line 5) and sets its location to the target URL `t`.

```
function comp(t)
{   tw = window.top;           // (4)
    afw = tw.frames["hide"];    // (5)
    afw.location=t;
}
```

The loading of the result page from the `fortune.pl` service is completely hidden. And the result page contains Javascript code to modify the DOM tree of the `show` frame.

The Perl code for `fortune.pl` is as follows.

```
#!/usr/bin/perl

my $xhtml_front =
'<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">';

$ft = '/usr/games/fortune';          ## (6)
print <<END;
Content-type: text/html

$xhtml_front
<head><title>Fortune</title>
<script type="text/javascript">
function act()
{   nd = document.getElementById("fortune"); // (7)
    code = nd.innerHTML;                    // (8)
    td = window.top.document;              // (9)
    f = td.getElementById("show");
    afd = f.contentDocument;
    bb = afd.getElementById("ans");
    bb.innerHTML=code;                      // (10)
}
</script></head>
<body onload="act()" id="fortune">
<p style="color: blue"> $ft </p>
```

```
</body></html>  
END  
exit;
```

The Perl program calls the UNIX program **fortune** (line 6) which generates a randomly selected "fortune cookie" message each time it is called. The "fortune" is sent in the `<body>` of the returned page. The Javascript function **act** is called when the returned page is loaded to obtain the HTML code in the `<body>` (lines 7-8) and place the HTML code in the **ans** element of the **show** frame (lines 9-10).

Due to security concerns, the Web service must be provided by the same Web server that supplied the **show** frame page. Only the Javascript code in such pages are allowed to access and modify the **show** frame page.

The list of files involved in this example are:

- **webservice.html**—the frameset
- **fortune.html**—the **show** frame page
- **empty.html**—the **hide** frame page
- **service.js**—Javascript included in **fortune.html**
- **fortune.pl**—Perl CGI script delivering the fortune cookie and Javascript code

The technique considered here can be applied together with the code experimenter (Section 10.19) to load different pages for experimentation.

10.21 For More Information

This chapter gets you started in standard-based DHTML. As DOM, Javascript, and CSS standards grow and evolve, and as browser compliance becomes more complete and wide spread, DHTML will be an increasingly more powerful and effective tool for delivering dynamic Web pages.

For more information on the W3C standards see www.w3c.org. For DOM bindings to the ECMA Script see

www.w3.org/TR/REC-DOM-Level-1/ecma-script-language-binding.html

For the Netscape implementation of DOM see the Gecko DOM reference

www.mozilla.org/docs/dom/domref/dom_shortTOC.html

For standard-based DHTML sample codes see, for example,

www.mozilla.org/docs/dom/samples/

[dmoz.org/Computers/Programming/Languages/
JavaScript/W3C_DOM/Sample_Code/](http://dmoz.org/Computers/Programming/Languages/JavaScript/W3C_DOM/Sample_Code/)

webfx.eae.net

10.22 Summary

DHTML is a technique that combines Javascript, CSS, and HTML. DOM is a W3C recommended standard API for accessing and modifying HTML and XML documents. DHTML with DOM results in powerful and cross-platform code.

Browsers support the DOM specified API and provides the required objects and methods for Javascript programming. The `window.document` object implements the `HTMLDocument` interface and gives you the root node of a Web page. Each element on this DOM tree corresponds to an HTML element in the source document and implements the `HTMLElement` interface, derived from `Element` which extends `Node`. The `document.getElementById` method is handy for obtaining the DOM object representing any HTML element with an assigned `id`. Starting from any node `el` on the DOM tree you can follow the child (`el.childNodes`, `el.firstChild`), parent (`el.parentNode`), and sibling (`el.nextSibling`, `el.previousSibling`)

Brooks/Cole book/January 28, 2003

relations to traverse the entire tree or any parts of it. You can also access and modify element attributes (`el.getAttribute(attr)`, `el.setAttribute(attr, value)`) and styles (`el.style.property`).

The DOM API allows you to systematically access, modify, delete and augment the DOM tree resulting in altered page display: `e.removeChild(node)`, `e.appendChild(node)`, `e.replaceChild(new, old)`, `e.insertBefore(new, node)`. New tree nodes can be created in Javascript with `document.createElement(tagName)`, `document.createTextNode(string)`.

When you combine event handling, including those generated by `window.setTimeout()`, and style manipulations, many interesting and effective dynamic effects can be achieved for your Web pages.

Accessing Web services with DHTML promises to allow developers to add value to Web pages by including information dynamically produced by various services on the Web.

Exercises

Review Questions

1. What is DHTML? Explain in your own words.
2. What are three important enabling technologies for standard-based DHTML?
3. What is DOM? the DOM tree? the most basic `Node` object?
4. Name important types of nodes on the DOM tree and describe the DOM tree for a Web page in detail.
5. Write down the Javascript code for obtaining the DOM node for an HTML element with a given *id* and for determining its node type.
6. What is the `nodeName` and `nodeValue` for an `HTMLElement`?
7. Describe the `HTMLElement` interface.

8. Explain the fields and methods in the `HTMLDocument` interface. Which object available by Javascript supports this interface?
9. How does Javascript modify the presentation style of an element on the DOM tree?
10. Compare the `window` object described in Section 10.18 and Section 9.12.
11. Explain the concept of Web service and the access of Web services with DHTML.

Assignments

1. Improve the Ex: **DomHello** in Section 10.2 and make the mouseover action also change the text “over the phrase” to “out of the phrase”. Test and see how well it works.
2. Modify the Ex: **DomCalc** in Section 10.6 and present a display in the form *string* = *result*.
3. Test the `normalize` method of `Node` (Section 10.7) to see how well it is supported by your browser.
4. Consider Ex: **DomDft** in Section 10.8). The traversal does not take comments into account. Modify the `dft` function to remedy this and test the traversal on pages that contain comments. (Hint: `Node.COMMENT_NODE` is 8.)
5. Consider the visual navigation of DOM tree (Section 10.9). Take (Ex: **DomNav**) and add a *tree display* of the table. As the user navigates the table, show the current position also on the DOM tree display.
6. Consider the guided form example in Section 10.11. Add the correctness check of email from Section 9.15) to it.
7. Follow the in-place fade-in example in Section 10.12 and write the code to achieve fade-out.

8. Take the "disappearing country code" problem described at the end of Section 10.16 and fashion a solution.
9. Add a *unmove* button to the TicTacToe program in Section 10.18 to take away the last move made.
10. Improve the TicTacToe program in Section 10.18 by adding the ability to play with the computer. (Hint: add move generation.)
11. Construct a pocket calculator using DHTML. Layout the LCD window and calculator buttons with a `table` and simulate the functions of the common calculator with `onclick` events on the buttons.



KARPAGAM ACADEMY OF HIGHER EDUCATION
COIMBATORE-21
Faculty of Engineering
Department of Computer Science and Engineering

Credit: 3	Possible Questions	R-2017
Subject Code : 16BECS702		Time :
Name of the Course :	B.E	Max Marks :
Title of the paper : INTERNET AND WEB TECHNOLOGY		Date :
Semester : VII		Session :

PART – A (20 x 1 = 20 Marks)
Answer ALL the questions
(Question Nos. 1 to 20 Online Examinations)

Q.No	PART – B (5x2 = 10 Marks) (2 ½ hours)	Bloom's	COs
	Answer ALL the questions ALL THE QUESTIONS CARRY EQUAL MARKS		
21.	Define CGI Client side applets.	[RE]	CO6
22.	What is meant by HTML tag emulation?	[RE]	CO4
23.	Outline the advantages of java script functions.	[UN]	CO2
24.	List out the properties of frameset tag.	[AN]	CO5
25.	Distinguish object and event model.	[AN]	CO6

PART– C (5x14 = 70 Marks)
Answer ALL the questions
ALL THE QUESTIONS CARRY EQUAL MARKS

26.	Explain in detail about HTML Tags with its attributes.	[UN]	CO4
	(OR)		
	Illustrate Internet protocols and its applications.	[UN]	CO2
27.	Discuss CGI script structure with example.	[CR]	CO6
	(OR)		
	Elaborate authorization and security facilities in CGI.	[CR]	CO6
28.	Construct XML syntax and structures rules and write down the components of XML file.	[CR]	CO5
	(OR)		
	Explain Scripting language and write a scripting program to add two integers.	[EV]	CO8
29.	Determine how DHTML used to develop the web pages?	[EV]	CO4
	(OR)		
	Explain the four way of adding style to a webpage with example.	[UN]	CO4
30.	Design and explain the servlet architecture with its functionality.	[CR]	CO9
	(OR)		
	Explain how the JSP requests handled? Illustrate with an example.	[UN]	CO9



KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE-21

Faculty of Engineering

Department of Computer Science and Engineering

Online Questions

Questions	opt1	opt2	opt3	opt4	Answer
session is instance of which class?	Session	HttpSession	HttpServletSession	ServletSession	HttpSession
2Which of the following do not supports JSP directly?	Weblogic Server	WebSphere Server	Tomcat Server	Apache HTTP Server	Apache HTTP Server
3Which of the following attributes are used in <jsp:include /> tag?	id, type	page, flush	type, class	type,page	page, flush
What is true about filters?	JSP Filters are used to intercept requests from a client before they access a resource at back end.	JSP Filters are used to manipulate responses from server before they are sent back to the client.	Both of the above.	None of the above.	Both of the above.
5What are cookies?	Cookies are text files stored on the client computer and they are kept for various information tracking purpose.	Cookies are binary files stored on the server computer and they are kept for various information tracking purpose.	Cookies are binary files stored on the client computer and they are kept for data storage purpose.	None of the above.	Cookies are text files stored on the client computer and they are kept for various information tracking purpose.

6Which of the following is true about session Attribute?	The session attribute indicates whether or not the JSP page uses HTTP sessions.	A value of true means that the JSP page has access to a builtin session object.	A value of false means that the JSP page cannot access the builtin session object.	All of the above.	All of the above.
Which of the following is true about Execution phase in JSP life cycle?	Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.	The _jspService() method of a JSP is invoked once per request and is responsible for generating the response for that request	The _jspService() method of a JSP generates response s to all seven of the HTTP methods ie. GET, POST, DELETE etc.	All of the above.	All of the above.
Which of the following is true about response.sendRedirect(url)?	The sendRedirect sends HTTP temporary redirect response to the browser.	Request data gets lost in case of sendRedirect.	Both of the above.	None of the above.	Both of the above.
Which of the following is true about <c:forEach > tag?	The <c:forEach > exists as a good alternative to embedding a Java for, while, or do-while loop via a scriptlet.	The <c:forEach > is used to iterate over a list of items in jsp.	None of the above.	Both of the above.	None of the above.

session is instance of which class?	ServletSession	Session	HttpSession	HttpServletSession	HttpSession
Which of the following code is used to delete an attribute from a HTTP Session object in servlets? x	session.removeAttribute(name)	B - session.setAttribute(name)	session.updateAttribute(name)	session.updateAttribute(name)	session.removeAttribute(name)
Which of the following code sends a cookie in servlet?	response.createCookie(cookie);	response.addCookie(cookie);	response.sendCookie(cookie);	None of the above	None of the above
How to create a cookie in servlet?	Use new operator.	Use request.getCookie() method	Use response.getCookie() method	None of the above	Use new operator.
Which of the following code encodes the specified URL for use in the sendRedirect method?	response.encodeRedirectURL(url)	request.encodeRedirectURL(url)	Header.encodeRedirectURL(url)	None of the above.	response.encodeRedirectURL(url)
Which of the following code retrieves any extra path information associated with the URL the client sent?	Header.getPathInfo()	response.getPathInfo()	request.getPathInfo()	None of the above.	request.getPathInfo()

Which of the following code is used to get locale in servlet?	new Locale()	request.getlocale()	response.getLocale()	None of the above.	request.getlocale()
Which of the following method can be used to get the value of form parameter?	request.getParameter()	request.getParameterValues()	request.getParameterNames()	None of the above.	request.getParameter()
When doPost() method of servlet gets called?	A POST request results from an HTML form that specifically lists POST as the METHOD.	The service() method checks the HTTP request type as POST and calls doPost() method.	Both of the above.	None of the above.	Both of the above.
Which of the following is true about <c:forEach > tag?	The <c:forEach > exists as a good alternative to embedding a Java for, while, or do-while loop via a scriptlet.	The <c:forEach > is used to iterate over a list of items in jsp.	Both of the above.	None of the above.	Both of the above.
Which of the following is true about response.sendRedirect(url)?	The sendRedirect sends HTTP temporary redirect response to the browser.	Request data gets lost in case of sendRedirect.	Both of the above.	None of the above.	Both of the above.
Which of the following is true about Execution phase in JSP life cycle?	Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the	The _jspService() method of a JSP is invoked once per request and is responsible for generating the response for	The _jspService() method of a JSP generates response	All of the above.	All of the above.

	_jspService() method in the JSP.	that request.	s to all seven of the HTTP methods ie. GET, POST, DELETE etc.		
Which page directive should be used in JSP to generate a PDF page?	contentType	generatePdf	typePDF	contentPDF	contentType
Which tag should be used to pass information from JSP to included JSP?	Using <%jsp:page> tag	Using <%jsp:param> tag	Using <%jsp:im port> tag	Using <%jsp:useBe an> tag	Using <%jsp:page > tag
Application is instance of which class?	javax.servlet.Appl ication	javax.servlet.Ht tpContext	javax.ser vlet.Cont ext	javax.servlet. ServletConte xt	javax.servl et.ServletC ontext
_jspService() method of HttpJspPage class should not be overridden.	TRUE	FALSE			TRUE
Which option is true about session scope?	Objects are accessible only from the page in which they are created	Objects are accessible only from the pages which are in same session	Objects are accessibl e only from the pages which are processin g the same request	Objects are accessible only from the pages which reside in same application	Objects are accessible only from the pages which are in same session
Which one is the correct order of phases in JSP life cycle?	Initialization, Cleanup, Compilation, Execution	Initialization, Compilation, Cleanup, Execution	Compilati on, Initializat ion, Executio n, Cleanup	Cleanup, Compilation, Initialization, Execution	Compilatio n, Initializatio n, Execution, Cleanup
"request" is instance of which one of the following classes?	Request	HttpRequest	HttpServletReques t	ServletReque st	HttpServletRequest

Which is not a directive?	include	page	export	useBean	export
Which is mandatory in tag?	id, class	id, type	type, property	type,id	id, class
Which of the following are the valid scopes in JSP?	request, page, session, application	request, page, session, global	response, page, session, application	request, page, context, application	request, page, session, application
Which of the following is an implicit object?	pageContext	servletContext	httpContext	sessionContext	pageContext
Which of the scripting of JSP not putting content into service method of the converted servlet?	Declarations	Scriptlets	Expressions	None of the above	Expressions
The difference between Servlets and JSP is the	translation	compilation	syntax	Both A and B	syntax
JSP includes a mechanism for defining or custom tags.	static attributes	local attributes	dynamic attributes	global attributes	dynamic attributes
Which describes best an EJB handle?	used to handle exceptions when accessing EJB objects	An EJB handle is used to store a reference to a specific EJB object	An EJB handle is part of the Home interface	An EJB handle is used for local references inside the EJB container	used to store a reference to a specific EJB object
Why DB connections are not written directly in JSPs?	Response is slow	Not a standard J2EE architecture	Load Balancing is not possible	Both B and C	Both B and C
How many jsp implicit objects are there and these objects are created by the web container that are available to all the	8	9	10	7	9

jsp pages?					
Why use Request Dispatcher to forward a request to another resource, instead of using a sendRedirect?	Redirects are no longer supported in the current servlet API	Redirects are not a cross-platform portable mechanism	The Request Dispatcher does not use the reflection API	The RequestDispatcher does not require a round trip to the client, and thus is more efficient and allows the server to maintain request state	The RequestDispatcher does not require a round trip to the client, and thus is more efficient and allows the server to maintain request state
Which attribute specifies a JSP page that should process any exceptions thrown but not caught in the current page?	The ErrorPage Attribute	The IsErrorPage Attribute	Both A & B	None of the above	The ErrorPage Attribute
Which http method send by browser that asks the server to get the page only?	get	post	option	put	get
JDBC is a interface, which means that it is used to invoke SQL commands directly	low-level	middle-level	higher-level	user	low-level
Which can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc. in JSP?	vs.Static HTML	vs.Server-Side Includes	vs.Pure Servlets	Vs.JavaScript	Vs.JavaScript

Dynamic interception of requests and responses to transform the information is done by	servlet container	servlet config	servlet context	servlet filter	servlet filter
What type of scriptlet code is better-suited to being factored forward into a servlet?	Code that deals with logic that is common across requests	Code that deals with logic that is vendor specific	Code that deals with logic that relates to database access	Code that deals with logic that relates to client scope	Code that deals with logic that is common across requests
This is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet.	Platform as a Service (PaaS)	Infrastructure as a Service (IaaS)	Software as a Service (SaaS)	None	Software as a Service (SaaS)
What is the initial contact point for handling a web request in a Page-Centric architecture?	A JSP page	A JavaBean	A servlet	A session manager	A JSP page
Java Soft provides JDBC product components as part of the java Developer's Kit (JDK)	three	two	four	single	three
How to send data in get method?	We can't	Through url	Through payload	None of these	Through url
What is the name of the organization helping to foster security standards for cloud computing?	Cloud Security Standards Working	Cloud Security Alliance	Cloud Security Watchdog	Security in the Cloud Alliance	Cloud Security Alliance

Which of the following statements is true regarding the scope of 'request' in JSP?	Objects with request scope are accessible from pages processing the same request where they were created	All references to the object shall be released after the request is processed; in particular, if the request is forwarded to a resource in the same run time, the object is still reachable	References to objects with request scope are stored in the request object	All of the above	All of the above
Which of the following code is used to get an attribute in a HTTP Session object in servlets?	session.getAttribute(String name)	session.setAttribute(String name)	session.updateAttribute(String name)	session.setAttribute(String name)	session.getAttribute(String name)
Which method is used to specify before any lines that uses the PrintWriter?	setPageType()	setContentType()	setContentType()	setResponseType()	setContent Type()
What are the functions of Servlet container?	Lifecycle management	Communication support	Multithreading support	All of the above	All of the above
What is bytecode?	Machine-specific code	Java code	Machine-independent code	None of the mentioned	Machine-independent code
Which object of HttpSession can be used to view and manipulate information about a session?	session identifier	creation time	last accessed time	All mentioned above	All mentioned above
Which JDBC driver Type(s) can be used in either applet or servlet code?	Both Type 1 and Type 2	Both Type 1 and Type 3	Both Type 3 and Type 4	Type 4 only	Both Type 3 and Type 4

Which of the following code retrieves the MIME type of the body of the request?	new MimeType()	request.getCo ntentType()	respons e.getCo ntentTyp e()	None of the above	request.ge tContentT ype()
RequestDispatche r object is used	to include other resources	to include an image	to include xml object	to include e- mailing response	to include other resources