

WEB TECHNOLOGY

NOTES

IT8501

UNIT I

9

Web Essentials: Clients, Servers, and Communication. The Internet-Basic Internet Protocols The World Wide Web-HTTP request message-response message-Web Clients Web Servers-Case Study. Markup Languages: XHTML . An Introduction to HTML History-Versions-Basic XHTML Syntax and Semantics-Some Fundamental HTML Elements-Relative URLs-Lists-tables-Frames-Forms-XML Creating HTML Documents Case Study.

UNIT II

9

Style Sheets: CSS-Introduction to Cascading Style Sheets-Features-Core Syntax-Style Sheets and HTML Style Rle Cascading and Inheritance-Text Properties-Box Model Normal Flow Box Layout-Beyond the Normal Flow-Other Properties-Case Study. Client- Side Programming: The JavaScript Language-History and Versions Introduction JavaScript in Perspective-Syntax Variables and Data Types-Statements-Operators-Literals-Functions-Objects-Arrays-Built-inObjects-JavaScript,Debuggers.

UNIT III

9

Host Objects : Browsers and the DOM-Introduction to the Document Object Model DOM History and Levels-Intrinsic Event Handling-Modifying Element Style-The Document Tree-DOM Event Handling-Accommodating Noncompliant Browsers Properties of window-Case Study. Server-Side Programming: Java Servlets- Architecture -Overview-A Servlet-Generating Dynamic Content-Life Cycle-Parameter Data-Sessions-Cookies- URL Rewriting-Other Capabilities-Data Storage Servelets and Concurrency-Case Study-Related Technologies.

UNIT IV

9

Representing Web Data: XML-Documents and Vocabularies-Versions and Declaration -Namespaces JavaScript and XML: Ajax-DOM based XML processing Event-oriented Parsing: SAX-Transforming XML Documents-Selecting XML Data:XPath-Template based Transformations: XSLT-Displaying XML Documents in Browsers-Case Study- Related Technologies. Separating Programming and Presentation: 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-CaseStudy-RelatedTechnologies.

UNIT V

9

Web Services: JAX-RPC-Concepts-Writing a Java Web Service-Writing a Java Web Service Client-Describing Web Services: WSDL- Representing Data Types: XML Schema-Communicating Object Data: SOAP Related Technologies-Software Installation-Storing Java Objects as Files-Databases and Java Servlets.

TEXT BOOK:

TOTAL=45 PERIODS

1. Jeffrey C.Jackson, "Web Technologies--A Computer Science Perspective", Pearson Education, 2006.

REFERENCES:

1. Robert. W. Sebesta, "Programming the World Wide Web", Fourth Edition, Pearson Education, 2007.
2. Deitel, Deitel, Goldberg, "Internet & World Wide Web How To Program", Third Edition, Pearson Education,2006.
3. Marty Hall and Larry Brown,"Core Web Programming" Second Edition, Volume I And II, Pearson Education , 2001.
4. Bates, "Developing Web Applications", Wiley, 2006.

UNIT I

WEB ESSENTIALS

Introduction:**Web Essentials:****Server:**

The software that distributes the information and the machine where the information and software reside is called the server.

- provides requested service to client
- e.g., Web server sends requested Web page

Client:

The software that resides on the remote machine, communicates with the server, fetches the information, processes it, and then displays it on the remote machine is called the client.

- initiates contact with server (-speaks first)
- typically requests service from server
- Web: client implemented in browser

Web server:

Software that delivers Web pages and other documents to browsers using the HTTP protocol

Web Page:

A web page is a document or resource of information that is suitable for the World Wide Web and can be accessed through a web browser.

Website:

A collection of pages on the World Wide Web that are accessible from the same URL and typically residing on the same server.

URL:

Uniform Resource Locator, the unique address which identifies a resource on the Internet for routing purposes.

Client-server paradigm:

The Client-Server paradigm is the most prevalent model for distributed computing protocols. It is the basis of all distributed computing paradigms at a higher level of abstraction. It is service-oriented, and employs a request-response protocol.

A server process, running on a server host, provides access to a service. A client process, running on a client host, accesses the service via the server process. The interaction of the process proceeds according to a protocol.

The primary idea of a client/server system is that you have a central repository of information—some kind of data, often in a database—that you want to distribute on demand to some set of people or machines.

The Internet:

- Medium for communication and interaction in inter connected network.
- Makes information constantly and instantly available to anyone with a connection.

Web Browsers:

- User agent for Web is called a browser:
 - o Internet Explorer
 - o Firefox

Web Server:

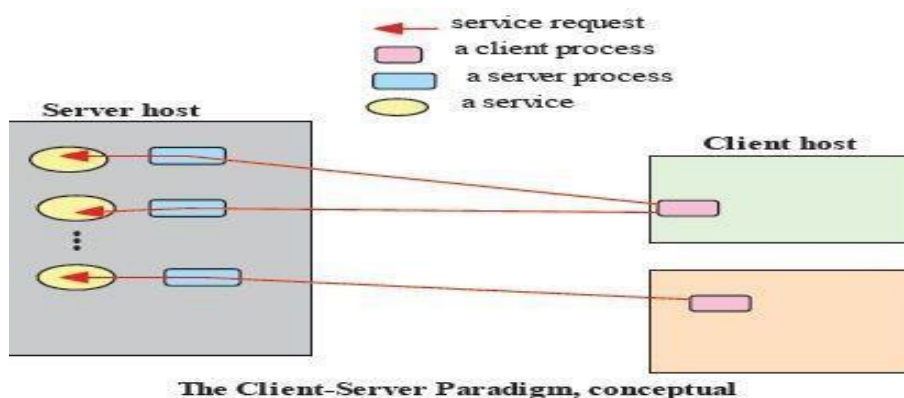
- Server for Web is called Web server:
 - o Apache (public domain)
 - o MS Internet Information Server

Protocol:

Protocols are agreed formats for transmitting data between devices.

The protocol determines:

- i. The error checking required
- ii. Data compression method used
- iii. The way the end of a message is signalled
- iv. The way the device indicates that it has received the message



Internet Protocol:

There are many protocols used by the Internet and the WWW:

- o TCP/IP
- o HTTP
- o FTP
- o Electronic mail protocols IMAP
- o POP

TCP/IP

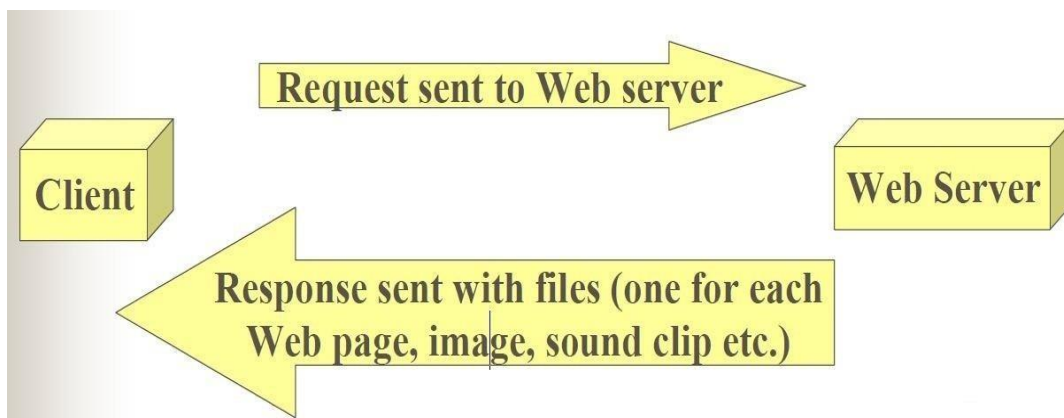
The Internet uses two main protocols (developed by Vincent Cerf and Robert Kahn)

Transmission control protocol (TCP): Controls disassembly of message into packets at the origin reassembles at the destination

Internet protocol (IP): Specifies the addressing details for each packet Each packet is labelled with its origin and destination.

Hypertext Transfer Protocol (HTTP)

- The hypertext transfer protocol (HTTP) was developed by Tim Berners-Lee in 1991
- HTTP was designed to transfer pages between machines
- The client (or Web browser) makes a request for a given page and the Server is responsible for finding it and returning it to the client
- The browser connects and requests a page from the server
- The server reads the page from the file system, sends it to the client and terminates the connection.



Electronic Mail Protocols:

- Electronic mail uses the client/server model
- The organisation has an email server devoted to handling email
 - o Stores and forwards email messages
- Individuals use email client software to read and send email
 - o (e.g. Microsoft Outlook, or Netscape Messenger)
- Simple Mail Transfer Protocol (SMTP)

- o Specifies format of mail messages
- Post Office Protocol (POP) tells the email server to:
 - o Send mail to the user's computer and delete it from the server
 - o Send mail to the user's computer and do not delete it from the server
 - o Ask whether new mail has arrived.

Interactive Mail Access Protocol (IMAP)

Newer than POP, provides similar functions with additional features.

o e.g. can send specific messages to the client rather than all the messages.

A user can view email message headers and the sender's name before downloading the entire message.

Allows users to delete and search mailboxes held on the email server.

The disadvantage of POP: You can only access messages from one PC.

The disadvantage of IMAP : Since email is stored on the email server, there is a need for more and more expensive (high speed) storage space.

World Wide Web: comprises software (Web server and browser) and data (Web sites).

Internet Protocol (IP) Addresses:

- Every node has a unique numeric address
- Form: 32-bit binary number
- New standard, IPv6, has 128 bits (1998)
- Organizations are assigned groups of IPs for their computers

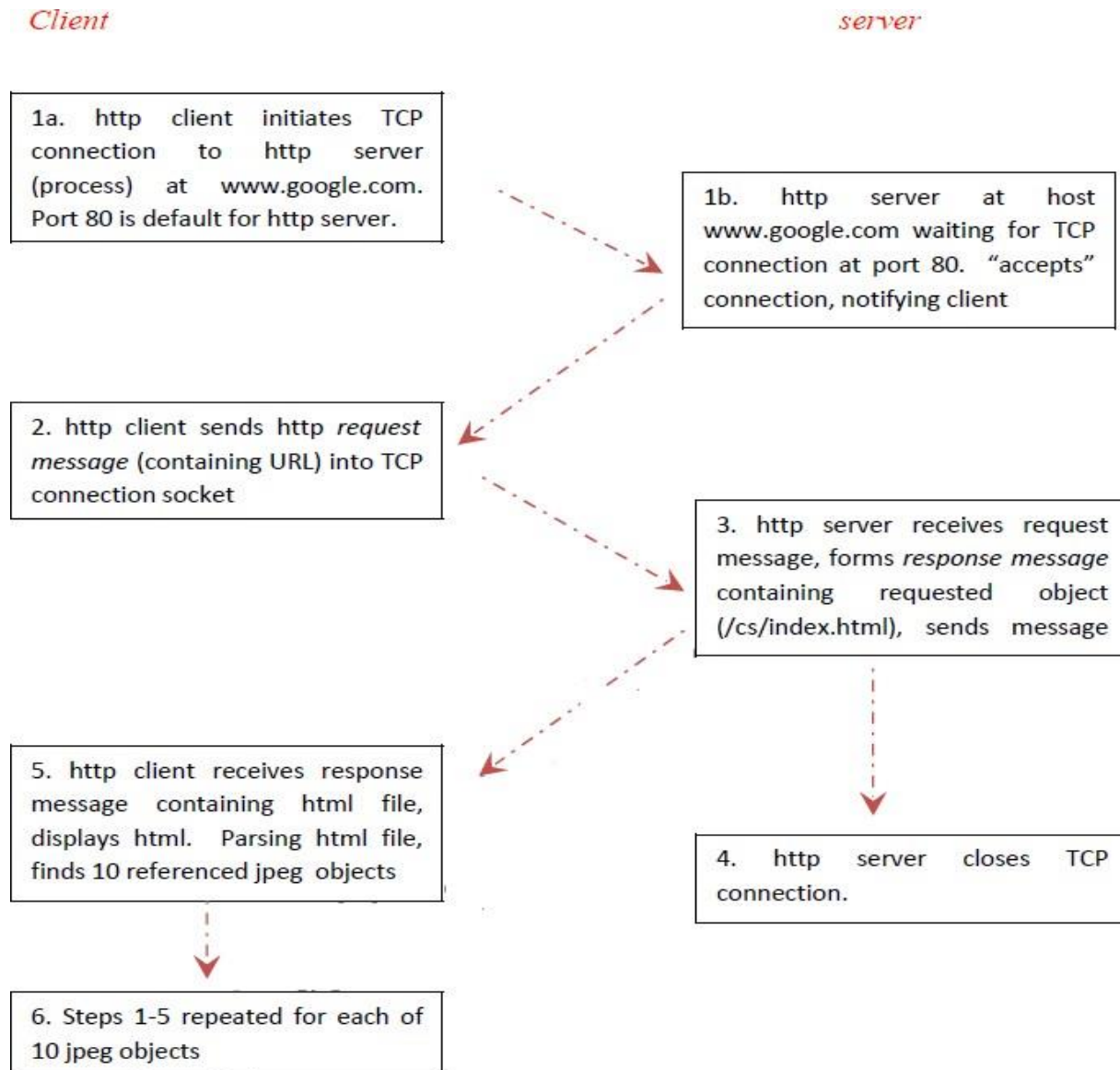
- Domain names

- Form: host-name. domain-names
- First domain is the smallest (Google)
- Last domain specifies the type of organization (.com)
- Fully qualified domain name - the host name and all of the domain names
- DNS servers - convert fully qualified domain names to IPs

HTTP:

- Hypertext Transfer Protocol (HTTP) is the communication protocol used by the Internet to transfer hypertext documents.
- A protocol to transfer hypertext requests and information between servers and browsers
- Hypertext is text, displayed on a computer, with references (hyperlinks) to other text that the reader can immediately follow, usually by a mouse HTTP is behind every request for a web document or graph, every click of a hypertext link, and every submission of a form.
- HTTP specifies how clients **request** data, and how servers **respond** to these requests.
- The client makes a request for a given page and the server is responsible for finding it and returning it to the client.
- The browser connects and requests a page from the server.
- The server reads the page from the file system and sends it to the client and then terminates the connection

HTTP Transactions

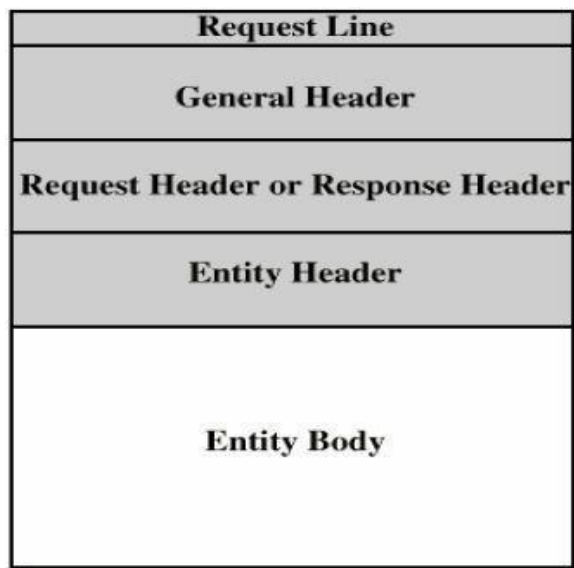


HTTP Message:

HTTP message is the information transaction between the client and server.

Two types of HTTP Message:

1. Requests
 - a. Client to server
2. Responses
 - a. Server to client

**Fields**

- Request line or Response line
- General header
- Request header or Response header
- Entity header
- Entity body

.10 Request Message:**Request Line:**

- A request line has three parts, separated by spaces
 - a *method* name
 - the local path of the requested resource
 - the version of HTTP being used
- A typical request line is:
 - GET /path/to/file/index.html HTTP/1.1
- Notes:
 - **GET** is the most common HTTP method; it says "give me this resource". Other methods include **POST** and **HEAD**. Method names are always uppercase
 - The path is the part of the URL after the host name, also called the *request URI*
 - The HTTP version always takes the form "**HTTP/x.x**", uppercase.

Request Header:

HTTP Request Headers

Header	Description
From	Email address of user
User-Agent	Client s/w
Accept File	File types that client will accept
Accept-encoding	Compression methods
Accept-Language	Languages
Referrer	URL of the last document the client displayed
If-Modified-Since	Return document only if modified since specified
Content-length	Length (in bytes) of data to follow

HTTP Request

The diagram shows an HTTP request line and its headers. The request line is `GET /msaleh/index.html HTTP/1.1`. Labels with arrows point to its parts: **Method** points to `GET`, **File name** points to `/msaleh/index.html`, and **HTTP version** points to `HTTP/1.1`. Below the request line is a blank line, labeled **Blank line**. The headers follow, labeled **Headers**. The headers are: `Host: staff.ifm.ac.tz`, `Connection: close`, `Accept: text/xml,text/html,text/plain,image/png,*/*`, `Accept-Language: en-gb,en`, `User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)`, `Accept-Charset: ISO-8859-1,utf-8;q=0.7,*`, `If-Modified-Since: Mon, 18 Sep 2006 22:57:19 GMT`, and `Referer: http://web-sniffer.net`. Below the headers is another blank line, labeled **Data – none for GET**.

.11 Response Message:

Response Line:

- A request line has three parts, separated by spaces
 - o the HTTP version,
 - o a *response status code* that gives the result of the request, and
 - o an English *reason phrase* describing the status code
- Typical status lines are:
 - o HTTP/1.0 200 OK or
 - o HTTP/1.0 404 Not Found
- Notes:
 - o The HTTP version is in the same format as in the request line, "**HTTP/x.x**".

o The status code is meant to be computer-readable; the reason phrase is meant to be human-readable, and may vary.

HTTP Request Header:

HTTP Response Headers	
Header	Description
Server	Server software
Date	Current Date
Last-Modified	Modification date of document
Expires	Date at which document expires
Location	The location of the document in redirection responses
Pragma	A hint, e.g., no cache
MIME-version	
Link	URL of document's parent
Content-Length	Length in bytes
Allowed	Requests that user can issue, e.g., GET

EXAMPLE

HTTP Response

The diagram shows an HTTP response structure with the following components labeled:

- HTTP version:** Points to "HTTP/1.0" in the first line.
- Status code:** Points to "200" in the first line.
- Reason phrase:** Points to "OK" in the first line.
- Headers:** Points to the block of lines starting with "Date:", "Server:", "Connection:", "Content-Type:", "ETag:", "Last-Modified:", and "Content-Length:".
- Data:** Points to the XML/HTML content block starting with "<?xml" and "<!DOCTYPE".

```

HTTP/1.0 200 OK
Date: Thu, 21 Sep 2006 22:06:05 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.10
Connection: close
Content-Type: text/html
ETag: "5d150-141c-450f244f"
Last-Modified: Mon, 18 Sep 2006 22:57:19 GMT
Content-Length: 5184

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
  
```

HTTP Method:

• HTTP method is supplied in the request line and specifies the operation that the client has requested.

Some common methods:

- Options
- Get
- Head
- Post
- Put
- Move
- Delete

Two methods that are mostly used are the GET and POST:

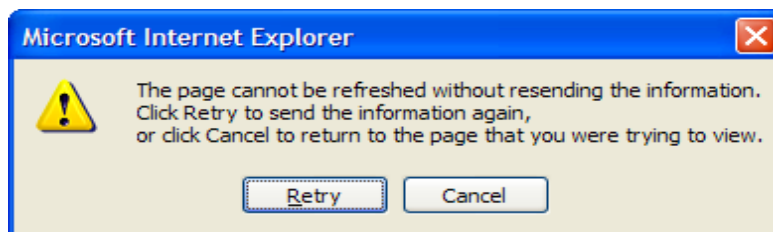
- o **GET** for queries that can be safely repeated
- o **POST** for operations that may have side effects (e.g. ordering a book from an on-line store).

The GET Method

- It is used to retrieve information from a specified URI and is assumed to be a safe, repeatable operation by browsers, caches and other HTTP aware components
- Operations have no side effects and GET requests can be re-issued.
- For example, displaying the balance of a bank account has no effect on the account and can be safely repeated.
- Most browsers will allow a user to refresh a page that resulted from a **GET**, without displaying any kind of warning
- Proxies may automatically retry **GET** requests if they encounter a temporary network connection problem.
- GET requests is that they can only supply data in the form of parameters encoded in the URI (known as a **Query String**) – [downside]
- Cannot be unused for uploading files or other operations that require large amounts of data to be sent to the server.

The POST Method

- Used for operations that have side effects and cannot be safely repeated.
- For example, transferring money from one bank account to another has side effects and should not be repeated without explicit approval by the user.
- If you try to refresh a page in Internet Explorer that resulted from a **POST**, it displays the following message to warn you that there may be side effects:



The **POST** request message has a content body that is normally used to send parameters and data

- The IIS server returns two status codes in its response for a **POST** request
 - o The first is **100 Continue** to indicate that it has successfully received the **POST** request
 - o The second is **200 OK** after the request has been processed.

HTTP response status codes

- Informational (1xx)
- Successful (2xx)
- Redirection (3xx)
 - o 301: moved permanently

- Client error (4xx)
 - o 403 : forbidden
 - o 404: Not found
- Server error (5xx)
 - o 503: Service unavailable
 - o 505: HTTP version not supported

HTTP

❖ Features

- Persistent TCP Connections: Remain open for multiple requests
- Partial Document Transfers: Clients can specify start and stop positions
- Conditional Fetch: Several additional conditions
- Better content negotiation
- More flexible authentication.

❖ Web Browsers :

- Mosaic - NCSA (Univ. of Illinois), in early 1993 First to use a GUI, led to Explosion of Web use Initially for X-Windows, under UNIX, but was ported to other platforms by late 1993
- Browsers are clients - always initiate, servers react (although sometimes servers require responses)
- Most requests are for existing documents, using Hypertext Transfer Protocol
- (HTTP) But some requests are for program execution, with the output being returned as a document.

Browser: A web browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.

❖ Web Servers:

- Provide responses to browser requests, either existing documents or dynamically Built documents.
 - Browser-server connection is now maintained through more than one request- Response cycle
 - All communications between browsers and servers use Hypertext Transfer Protocol
 - Web servers run as background processes in the operating system.
 - Monitor a communications port on the host, accepting HTTP messages when they appear
- All current Web servers came from either
1. The original from CERN
 2. The second one, from NCSA
- Web servers have two main directories:
 1. Document root (servable documents)
 2. Server root (server system software)
 - Document root is accessed indirectly by clients
 - Its actual location is set by the server Configuration file
 - Requests are mapped to the actual location
 - Virtual document trees
 - Virtual hosts
 - Proxy servers
 - Web servers now support other Internet protocols

- Apache (open source, fast, reliable)
- IIS
- Maintained through a program with a GUI interface.

❖ **Markup Language:**

Mark-up Language is used to identify elements of a page so that a browser can render that page on your computer screen.

Content to be displayed is —marked up or tagged to tell the browser how to display it.

A **markup language** is a set of characters or symbols that define a document's logical structure or how a document should be printed or displayed.

❖ **Hyper Text Markup Language:**

Hyper Text Mark-up Language, the language used to create documents on the World Wide Web.

· HTML is a collection of styles (indicated by mark-up *tags*) that define the various *elements* of a World Wide Web document.

· HTML is based on an older language called **Standard Generalized Markup Language**, or **SGML**, which defines the data in a document independently of how the data will be displayed.

· HTML document can be read and displayed by many different browsers running on different types of computers – platform independent!

· HTML defines the structure and layout of the elements on a Web page with a variety of *tags*.

· Each tag may have a set of *attributes* that modify the appearance or layout of the visual element *contained* by the tag.

· HTML is a plain-text file that can be created using a text editor like Notepad.

· HTML is a programming language that allows you to tell the browser what you want it to display and how you want it to be displayed, in simple terms, it is a Webpage.

· HTML there are certain markers, like commands, that tell the Browser what to do, these are called tags. By using tags you can have tables, fonts, colors, pictures, links, and almost anything you can think up, and experimentation with tags can lead to some pretty cool WebPages.

HTML History:

· The first version of HTML was created using the **Standard Generalized Mark up Language (SGML)**.

· In the early years of HTML, Web developers were free to define and modify HTML in whatever ways they thought best.

· Competing browsers introduced some differences in the language. The changes were called **extensions**.

· A group of Web developers, programmers, and authors called the **World Wide Web Consortium**, or the **WC3**, created a set of standards or specifications that all browser manufacturers were to follow.

· The **WC3** has no enforcement power.

· The recommendations of the **WC3** are usually followed since a uniform approach to Web page creation is beneficial to everyone.

Comparison of HTML v e r s i o n s :

HTML 3	HTML 4	HTML 5
<code><body></code> <code><table></code> <code><tr></code> <code><td>header</td></code> <code></tr></code> <code><tr></code> <code><td>nav</td></code> <code></tr></code> <code><tr></code> <code><td>left </td></code> <code><td> right </td></code> <code></tr></code> <code><tr></code> <code><td>footer</td></code> <code></tr></code> <code></table></code> <code></body></code>	<code><body></code> <code><div id="header">...</div></code> <code><div id="nav">...</div></code> <code><div class="right"></code> <code>...</code> <code></div></code> <code><div id="left">...</div></code> <code><div id="footer">...</div></code> <code></body></code>	<code><body></code> <code><header>...</header></code> <code><nav>...</nav></code> <code><right></code> <code>...</code> <code></right></code> <code><left>...</left></code> <code><footer>...</footer></code> <code></body></code>

HTML Basics:

HTML is primarily composed of two types of markup:

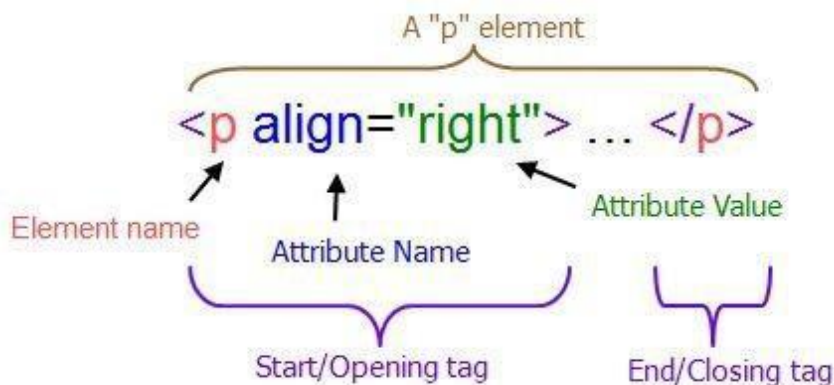
- Elements or tags
 - o `<html></html>`
- Attributes that modify an element

HTML Elements

- Elements are the fundamental building blocks of HTML.
- They are the tags that tell the browser what the enclosed text is.

HTML Tags:

- Container Tags
 - o `<Begin formatting>some text</End formatting>`
 - o For example: `<B>some bold text</B>`
`<H1>some heading </H1>`
- Empty Element Tags
 - o For example `<HR>` `<BR>`
- Comment Tag
 - o `<!-- Hi, I'm a comment. -->`
 - o Use them document complicated layouts!



- Case insensitive
- Unrecognised tags are simply ignored by browser!!
- Container tags must be nested!!
- As a text document, your HTML in Notepad will contain *elements*, such as headers, titles, paragraphs, etc.
- These elements must be denoted in your script, which is done using *tags*
- HTML tags consist of a left angle bracket (`<`), a name, and a right angle bracket (`>`)
- For example: `<title>`
- Tags must also close. To do so, you incorporate a slash (`/`). A starting and ending tag would be: `<title></title>`.

Attributes:

- You can add attributes to tags to enhance your page.
- Added attributes go inside the brackets of the opening tag(example: <p align=center> would center the paragraph)

About HTML file Structure:

- HTML files **.htm** or **.html** extensions
- Name your files to describe their functionality file name of your home page should be **index.html**

Common Tags (Elements):

- Always include the <HTML>...</HTML> tags
- Comments placed inside <!--...--> tags
- HTML documents
 - o **HEAD** section.
 - Info about the document.
 - Info in header not generally rendered in display window.
 - **TITLE** element names your Web page.
 - o **BODY** section
 - Page content
 - Includes text, images, links, forms, etc.
 - Elements include backgrounds, link colors and font faces
 - **P** element forms a paragraph, blank line before and after

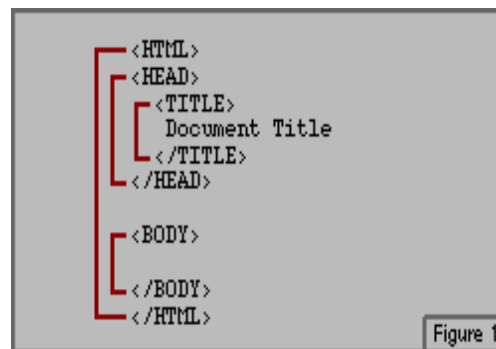
Structure of HTML Tag (Element) Alignment

Figure 1

Text Styling

- Underline style<U>...</U>
- Align elements with **ALIGN** attributeright, left or center
- Close nested tags in the reverse order from which they were opened
- Emphasis (italics) style...
- Strong (bold) style...
- and <I> tags deprecated Overstep boundary between content and presentation
- Logical Styles:

- o , a add emphasis to text
- o <BIG>, <SMALL> a increase or decrease text size
- o <SUB>, <SUP> a subscript or superscript
- Physical styles:
 - o , <I>, <U> a Bold, Italics, and Underline text
 - o
 - o E.g.,

eXtensible Mark up Language (XHTML) :

- To describe the general form and layout of documents
- An XHTML document is a mix of content and controls
- In *XHTML* tag names, attributes, and values are case sensitive and values must be enclosed by double quotes.
- In *XHTML* all elements must have both beginning and ending tags.
- Controls are tags and their attributes
 - o Tags often delimit content and specify something about how the content should be arranged in the document.
 - o Attributes provide additional information about the content of a tag.
- *Tools for creating XHTML documents*
 - o XHTML editors - make document creation easier
 - o Shortcuts to typing tag names, spell-checker,
 - o WYSIWYG XHTML editors
- *Plug ins*
 - o Integrated into tools like word processors,
 - o effectively converting them to WYSIWYG
 - o XHTML editors
- *Filters*
 - o Convert documents in other formats to XHTML

Advantages of both filters and plug-ins:

- Existing documents produced with other tools
- can be converted to XHTML documents
- Use a tool you already know to produce XHTML

Disadvantages of both filters and plug-ins:

- XHTML output of both is not perfect - must be fine tuned
- XHTML may be non-standard
- You have two versions of the document, which are difficult to synchronize

Relative URL

HTML Link:

To create a link to a resource identifiable by a URL

- o href: specify a URL of the target resource
- o target: specify where to display the target document
e.g.: `<a href="index.htm" target="_blank">Home</a>`

Open the document "index.htm" in a new browser window

Can also be used to create an anchor within a document

- o name: specify the anchor name
e.g.: `<a name="chap1"></a><h2>Chapter 1</h2>`

The above anchor can be referred to in a URL as

`<a href="http://host/file.html#chap1">Chapter 1</a>`

URL (Uniform Resource Locator) in HTML:

URL is used to create a link in a web document.

Two Types of URL:

1. Absolute URL

- Absolute URL contains all the information necessary to identify files on the internet (Example: in postal service, for sending letter to the destination it necessary to have full information like, name, address, city etc.,)

- Likewise, an absolute URL contains the protocol, hostname, filename, which are all essential to link the web document.

- Example:

- i. `http://google.com/index.html`

Relative URL

- Relative URL contains the only the folder name and filename or even just the file name.
- Browser does not need protocol address or server name to identify the file in the web.
- Relative URL to refer the documents in relation to the originating document.

Two types of Relative URL:

i. Document – Relative URL.

- Document Relative URL is used to relate the HTML document in the same folder, just providing the name of the file (ex: Index.html).

- It contains only the folder name and file name.

ii. Server – Relative URL

- Server relative URL is relative to the server root.

- It contains the part of URL and relating to the host name.

Anchors and Links

This section discusses the A tag which is used to define anchors (places in a document that can be linked to) and also to create links.

- **A**

A - (anchor or link)

The A tag lets you define anchors and links. An anchor defines a place in a document. A link displays a hypertext link that the user can click to display an anchor or a document.

A as anchor

An anchor identifies a place in an HTML document. To indicate that an <A> tag is being used as an anchor, specify the NAME attribute.

Note that you can also use the ID attribute of any tag to identify that tag as an anchor, as discussed in UNIVERSAL Attributes.

Do not nest an anchor within another A tag.

Syntax

```
<A  
  NAME="anchorName"  
>  
..  
</A>
```

Example

```
<A NAME=section2>  
<H2>A Cold Autumn Day</H2></A>
```

If this anchor is in a file called "nowhere.htm," you could define a link that jumps to the anchor as follows:

```
<P>Jump to the second section <A HREF="nowhere.htm#section2">  
A Cold Autumn Day</A> in the mystery "A man from Nowhere."
```

A as link

- A hypertext link is a piece of content that the user can click to invoke an action.
- The most common actions are scrolling to a different place in the current document and opening a new document.
 - A hypertext link can contain text and/or graphics.
 - To define a hypertext link, use the <A> tag with an HREF attribute to indicate the start of the hypertext link, and use the tag to indicate the end of the link.
 - When the user clicks any content between the <A HREF> and tags, the link is activated.
 - The value of the HREF attribute must be a URL.
 - If you want the link to open a new document, the value of HREF should be the URL for the destination document.
 - If you want to scroll the current document to a particular place, the value of HREF should be the name of the anchor to which to scroll, preceded by the # sign.
 - If you want to open another document at an anchor, give the URL for the document, followed by #, followed by the name of the anchor.
 - If you want the destination document or anchor to open in a separate browser window, supply the name of the window as the value of the TARGET attribute.
 - If the named window does not already exist, a new window opens.

- The link can also do actions other than opening and scrolling documents. It can send mail messages, point to files located on FTP servers, run any arbitrary JavaScript code, open local files, point to a gopher server, or read news groups.

- To do any of these actions, you specify an appropriate kind of URL, such as a `mailto` URL to send a mail message or a news URL to read a news group.

- Most browsers display hypertext links in a color different from that of the rest of the document so that users can easily identify them.

- You can also define actions that occur when the mouse cursor enters or leaves the region containing the link by specifying `onMouseOver` and `onMouseOut` event handlers for the link.

- Additionally, you can specify an `onClick` event handler that determines the action to occur when the user clicks a link.

- A link that has not been clicked is called an unvisited link.

- A link that has been clicked is known as a visited or followed link.

- A link that is in the process of being clicked is an active link.

Syntax

```
<A  
  HREF="location"  
  ONCLICK="clickJScript"  
  ONMOUSEOUT="outJScript"  
  ONMOUSEOVER="overJScript"  
  TARGET="windowName"  
>  
...  
</A>
```

HREF="location" specifies a destination URL for the link. The most common value here is a document name or an anchor. To specify a document to open, provide the URL for the document, either as an absolute or relative URL.

An example of an absolute URL is:

```
HREF="http://www.chennaikavigal.com/index.html"
```

An example of a relative URL is:

```
HREF="products/doc1.html"
```

To scroll the current document to an anchor, give the name of the anchor preceded by the pound (#) sign. For example:

```
HREF="#anchor1"
```

To open a different document at an anchor, give the URL for the document, followed by the # sign followed by the name of the anchor. For example:

```
HREF="products/doc1.html"
```

The **HREF** attribute can also be a URL that sends a message, points to files located on an FTP server, runs arbitrary JavaScript code, opens local files, points to a gopher server, or reads news groups.

ONCLICK="clickJScript" specifies JavaScript code to execute when a user clicks the link. If you supply a value for the ONCLICK attribute, the specified action overrides the default link behavior.

ONMOUSEOUT="outJScript" specifies JavaScript code to execute when a user moves the mouse cursor out of the link or anchor.

ONMOUSEOVER="overJScript" specifies JavaScript code to execute when a user moves the mouse pointer over the image or link text.

TARGET="windowName" specifies a window in which to show the destination document (if the link's action is to scroll or open a document). If the named window is not already open, a new window with that name opens.

Special target values are:

blank opens the destination document in a new unnamed window.

parent opens the destination document in the parent window of the one displaying the current document.

self opens the destination document in the same window as the one in which the link was clicked.

top opens the destination document in the full body of the current window. This value can be used to ensure that the destination document takes over the full window even if the original document was displayed in a frame.

Example

```
<P>You can find all the latest news from Chennai Kavigal at  
<A HREF="http://www.chennaikavigal.com/index.html">Chennai Kavigal's  
Home Page</A>.
```

HTML List:

Lists can add a lot to a WebPage, there are different types of lists and different types of bullets or numbers that can be used.

You can have an ordered list (a, b, c, etc.) unordered lists (with bullets) or you can have definition lists.

This section describes the tags for displaying lists:

DIR, DL, DT, DD, MENU, OL, UL, LI

DIR - (directory list)

The DIR tag is intended to display a list of short items such as in a directory listing. In practice, the DIR tag usually does the same thing as the UL (unordered list) tag.

To mark the individual items within the list, use the LI tag. A single list item can contain additional paragraphs, marked with the P tag.

This tag is being deprecated; that is, it has become obsolete.

Syntax

<DIR>

...

</DIR>

Example

<P>The directory structure is:</P>

<DIR>

PRODUCTS

<DIR>

SHAKTHI.htm

TYPING TUTOR.htm

</DIR>

MULTIMEDIA

<DIR>

KURALAMUDHU.htm

ARIVODU VILAYADU.htm

</DIR>

<DIR>

DL - (definition list)

The DL tag encloses a definition list. A definition list contains terms, which are defined with the DT tag, and definitions, which are defined with the DD tag. By default, Nadodi aligns terms on the left and indents each definition on a new line. However, you can use the COMPACT attribute to display a definition on the same line as the preceding term, if it fits on one line.

The intent of a definition list is to display lists of terms and their corresponding definitions, such as in a glossary.

Syntax

<DL COMPACT>

...

</DL>

COMPACT compacts the definition list by placing the term defined by the DT tag on the same line as the definition defined by the DD tag, provided the term is short enough.

Definition List Example

The following example defines six HTML terms.

<DL>

<DT>HTML

<DD>This tag marks a text file as an HTML document.

<DT>HEAD

<DD>This tag encloses the heading for the HTML document.

<DT>BODY

<DD>This tag displays the body for the HTML document.

<DT>DL

<DD>This tag displays a definition list in the HTML document.

<DT>DT

<DD>This tag displays a term in a definition list.

<DT>DD

<DD>This tag displays a definition description.

</DL>

DT - (definition term)

The DT tag specifies a term in a definition list. The DT tag must be used in a DL tag. Usually a DT tag is followed by a DD tag that describes the term in the DT tag.

The DT tag does not require a closing tag.

Syntax <DT>

Used Within <DL>

DD – (Definition description)

The DD tag displays a definition description in a definition list. The DD tag must be used within a DL tag and usually follows immediately after a DT tag that indicates the term being defined. The DD tag does not require a closing tag.

See DL for more details and an example.

Syntax <DD>

Used Within <DL>

MENU - (list of simple items)

The MENU tag displays a list of simple items. This tag works just like the UL tag. Use the LI tag to designate the individual menu items.

This tag is being deprecated because it has become obsolete in favor of the UL tag.

Syntax

<MENU>

...

</MENU>

Example

The following example creates a list of three short items:

```
<P>Menu Example:</P>
```

```
<MENU>
```

```
<LI> Windows 9x
```

```
<LI> Windows NT
```

```
<LI> Windows 2000
```

```
</MENU>
```

1. Unordered List:

- In an unordered list each item has the same leading symbol.
- The item in the list will be rendered in the same order that they are entered.
- `<ul>` tag is used to unordered the list.
- `<li>` tag is used to list the items.

Syntax

1. `<UL TYPE="CIRCLE"|"DISC"|"SQUARE">`

TYPE defines the type of bullet used for each list item. The value can be one of the following:

CIRCLE specifies a hollow bullet.

DISC specifies a solid round bullet (Nadodi's default).

SQUARE specifies a square bullet.

Example

```
<ul type=||disc||>
<li>Item 1</li>
<li>Item 2</li>
</ul>
<ul type="square">
<li>Item 1</li>
<li>Item 2</li>
</ul>
<ul type="circle">
<li>Item 1</li>
<li>Item 2</li>
</ul>
```

2. Ordered List:

- In an ordered list, each item has a sequential leading symbol.
- The browser assigns this sequence automatically.
- `<ol>` tag is used to ordered the list.
- `<li>` tag is used to list the item.

Syntax

```
<OL
START="value"
TYPE="A"|"a"|"I"|"i"|"1"
>
...
</OL>
```

START="value" indicates the starting number for the list. The number must be a positive integer.

TYPE defines the type of numbering sequence used for each list item.

The value can be one of the following:

- `A` specifies a sequence of uppercase letters
- `a` specifies a sequence of lowercase letters
- `I` specifies a sequence of uppercase Roman numerals
- `i` specifies a sequence of lowercase Roman numeral
- `1` specifies a sequence of numbers.

Example

```
<ol>
```



```
<li>Item 1</li>
<li>Item 2</li>
</ol>
<ol type="A" start="5">
<li>Item 1</li>
<li>Item 2</li>
</ol>
<ol type="i" start="10">
<li>Item 1</li>
<li>Item 2</li>
</ol>
```

3. Definition List

- A definition list is a sequence of terms and description of these terms.
- <dl> tag is used to implement definition list.
- <dt> tag is used for defining the item.
- <dd> tag is used for defining the description.

LI - (list item)

The LI tag indicates an itemized element, which is usually preceded by a bullet, a number, or a letter. The LI tag is used inside list elements such as OL (ordered list) and UL (unordered list).

A single itemized element can contain other tags such as the P tag.

The LI tag does not require a closing tag.

Syntax

```
<LI
TYPE="DISC"|"CIRCLE"|"SQUARE"|"A"|"a"|"I"|"i"|"1"
VALUE="number"
>
```

TYPE specifies the type of symbol or numbering sequence to use before each item.

- **DISC** specifies a solid bullet.
- **CIRCLE** specifies a hollow bullet.
- **SQUARE** specifies a square bullet.
- **A** specifies a sequence of uppercase letters.
- **a** specifies a sequence of lowercase letters.
- **I** specifies a sequence of uppercase Roman numerals.
- **i** specifies a sequence of lowercase Roman numeral.
- **1** specifies a sequence of numbers.

The values DISC, CIRCLE, and SQUARE can be used in unordered lists, while the values A, a, I, i, and 1 can be used in ordered lists that have a numerical sequence.

VALUE="number" indicates the starting number for an item in an ordered list. This attribute is valid only in an ordered list. See OL for information on the types of numbering available.

Used Within DIR, DL, OL, UL, MENU

Example

```
<UL>
```

```
<LI>Identify items in a numbered list
<LI>Identify items in an unordered list
<LI>Identify items in a directory list
<LI>Identify items in a menu
</UL>
```

HTML Table:

- Tables in HTML are easy to program if you know what you are trying to do.
- Tables look like the charts present on many of these slides.
- Tables in browsers by default have no borders, if you want to see the edges of your table, you have to place a border command inside the table tag.
- You can also use tables to format pages.
- Inside tables, the text is always the default of the browser (black and small), if you want to change this font, you must add a `<font color=-color|| and/or size=-x||>` inside each cell to make the change.

- **TABLE**
- **CAPTION**
- **TR**
- **TD**
- **TH**

1.14 TABLE - (table): The TABLE tag defines a table. Inside the TABLE tag, use the TR tag to define rows in the table, use the TH tag to define row or column headings, and use the TD tag to define table cells.

The TABLE tag can also contain a CAPTION tag, which specifies the caption for the table.

You can specify the width of the border surrounding the table and the default background color of the table. (Individual rows and cells in the table can have their own background color.) You can use the CELLSPACING attribute to specify the distance between cells in the table and the CELLPADDING attribute to specify the distance between the border and content of every cell. If you specify the width and height of the table, the browser will do its best to make the table fit the specified dimensions, but in some cases this may not be possible. For example, if the table contains cells that contain non-wrapping long lines, the table may not fit in a specified width.

Syntax

```
<TABLE
ALIGN="LEFT|RIGHT"
BGCOLOR="color"
BORDER="value"
CELLPADDING="value"
CELLSPACING="value"
HEIGHT="height"
WIDTH="width"
COLS="numOfCols"
HSPACE="horizMargin"
VSPACE="vertMargin"
```

```
>  
...  
</TABLE>
```

ALIGN specifies the horizontal placement of the table.

- **LEFT** aligns the table on the left (the default). The content following the table flows to the right of the table.
- **RIGHT** aligns the table on the right. The content following the table flows to the left of the table.
- **CENTER** aligns the table in the center. Content does not flow on either side.

BGCOLOR="color" sets the color of the background for the table. This color can be overridden by a **BGCOLOR** tag in the TH, TR, or TD tags.

BORDER="value" indicates the thickness, in pixels, of the border to draw around the table. Give the value as an integer. A value of 0 means the table has no border. You can also supply the **BORDER** attribute without specifying a value for it to indicate that the table has a border of the default thickness.

CELLPADDING="value" determines the amount of space, in pixels, between the border of a cell and the contents of the cell. The default is 1.

CELLSPACING="value" determines the amount of space, in pixels, between individual cells in a table. The default is 2.

HEIGHT="height" specifies the height of the table. The default is the optimal height determined by the contents of each cell. The height value can be a number of pixels, given as an integer, or a percentage of the height of the page or parent element, given as an integer followed by the percent sign. The table is scaled to fit the specified height and width.

WIDTH="width" defines the width of the table. The default is the optimal width determined by the contents of each cell.

The width value can be a number of pixels, given as an integer, or a percentage of the width of the page or parent element, given as an integer followed by the percent sign. The table is scaled to fit the specified height and width.

COLS="numOfCols" indicates how many virtual columns of equal width fit in the width of the window. Each actual column in the table occupies a virtual column. You would typically set the **COLS** attribute to be equal to the number of columns in the table to indicate that all the columns in the table have the same width.

If the **WIDTH** attribute is supplied, the **COLS** attribute indicates how many virtual columns fit in the specified width. If the **WIDTH** attribute is not supplied, the **COLS** attribute indicates how many virtual columns fit in the current window or frame. Each column in the table occupies one of the virtual columns. Suppose that the **WIDTH** attribute is "80%" and the **COLS** attribute is 4.

In this case, each virtual column takes up 20% of the width of the window. Each actual column in the table occupies a virtual column, so it occupies 20% of the width of the window, so long as the table has from 1 to 4 columns inclusive. Note, however, that if the minimum width needed to display the contents of an actual column is greater than the width of a virtual column, then the width of the column is expanded to fit its contents.

If the table has more actual columns than the **COLS** value, then the columns in excess of the **COLS** value are displayed in the minimum width required to fit their contents, and the other columns divide the remaining space equally between them.

For example, suppose the table has 4 columns, the **WIDTH** attribute is "80%", and the **COLS** value is 3. What happens here is that the table takes up 80% of the width of the window. The fourth column uses

the minimum width necessary to display the contents of the column. The other 3 columns divide the remaining width of the table equally between them.

HSPACE="horizMargin" specifies the distance between the left and right edges of the table and any surrounding content.

VSPACE="vertMargin" specifies the distance between the top and bottom edges of the table and any surrounding content.

Example 1. A Simple Table.

The following example creates a three-column, four-row table, with a yellow background. The caption "Tables are as easy as one, two, three" is displayed at the bottom of the table.

```
<TABLE BORDER CELLPADDING="8" CELLSPACING="4"
BGCOLOR=yellow>
<TR><TH> English </TH><TH> Spanish </TH><TH> German
</TH></TR>
<TR><TD> one </TD><TD> uno </TD><TD> ein </TD></TR>
<TR><TD> two </TD><TD> dos </TD><TD> zwei </TD></TR>
<TR><TD> three </TD><TD> tres </TD><TD> drei </TD></TR>
<CAPTION ALIGN="BOTTOM"> <B>Table 1</B>: Tables are as easy as
one, two, three
</CAPTION>
</TABLE>
```

Example 2: A More Complex Table.

The following example creates a four-column table. Each row has a different background color. The last row contains only two cells, which both span two rows, and the second cell spans three columns.

```
<TABLE CELLPADDING=3 CELLSPACING=6 BORDER=2>
<CAPTION ALIGN=TOP>
<BIG><BIG>Furniture Mart's Top Selling Furniture
</BIG></BIG>
</CAPTION>
<!-- heading row -->
<TR BGCOLOR=#CCCCFF>
<TH>NAME</TH>
<TH>SKU</TH>
<TH>PRICE</TH>
<TH>DESCRIPTION</TH>
</TR>
<!-- end of heading row -->
<!-- furniture rows -->
<TR BGCOLOR=#DDEEAA>
<TH>Harriet Smythe Armchair</TH>
<TD>100584</TD><TD>$2150</TD><TD>description goes here</TD>
<TR>
<TR BGCOLOR=#CCFFCC>
<TH>St. Michael Sofa</TH>
<TD>100789</TD><TD>$5000</TD><TD>description goes here</TD>
```

```

<TR>
<TR BGCOLOR=#BDDDF>
<TH>Variety of prints</TH>
<TD>.</TD><TD>$100 to $5000</TD><TD>description goes here</TD>
<TR>
<!-- more furniture rows go here -->
<!-- last row has cells that span rows and columns -->
<TR BGCOLOR=CYAN>
<TH ALIGN=CENTER VALIGN=MIDDLE ROWSPAN=2>
<FONT SIZE=+3>JULY SALE!!</FONT></TH>
<TD ROWSPAN=2 COLSPAN=3 ALIGN=CENTER>
<FONT SIZE=+1>
Don't miss our annual July sale.
All these prices will be slashed by 50%!!!
But on Aug 1, they go back up, so don't be late!!
</FONT>
</TD>
<TR>
</TABLE>

```

CAPTION - (table caption)

The CAPTION tag defines a caption for a table. Place the CAPTION tag within the TABLE tag but not inside the TD or the TR tags, which indicate table cells and table rows respectively.

Navigator 1.1.

Syntax

```
<CAPTION ALIGN="BOTTOM"|"TOP">...</CAPTION>
```

ALIGN specifies the placement of the caption within a table.

- BOTTOM places the caption at the bottom of the table.
- TOP places the caption at the top of the table. TOP is the default.

Used Within TABLE

Example See example 1. Simple Table.

TR - (table row) The TR tag specifies a table row. Use the TR tag inside a TABLE tag. The TR tag can contain TH tags, which indicate table headings, and TD tags, which indicate table cells.

Syntax

```

<TR
ALIGN="CENTER|LEFT|RIGHT"
VALIGN="BASELINE|BOTTOM|MIDDLE|TOP"
BGCOLOR="color"
>
...
</TR>

```

ALIGN specifies the horizontal placement of the table:

- CENTER centers the table .
- LEFT aligns the table to the left (the default).
- RIGHT aligns the table to the right.

VALIGN specifies the vertical placement of the content in the cell:

- BASELINE aligns the content with the cell's baseline.

- BOTTOM aligns the content with the cell's bottom.
- MIDDLE centers the content within the cell (the default).
- TOP aligns the content with the cell's top.

BGCOLOR="color" sets the default color of the background of the table row. Table cells defined with the TD tag inside the row can set their own background color.

Used within TABLE

Example See example 1. Simple Table.

TD - (table data)

The TD tag specifies text in a cell in a table. Use the TD tag inside a TR tag inside a TABLE tag.

You can set the background color of a cell by specifying its BGCOLOR attribute. For each cell, you can use the COLSPAN and ROWSPAN attributes to specify how many columns and rows the cell spans.

To specify the distance between cells, set the CELLSPACING attribute in the TABLE tag. To specify the distance between the borders of each cell and its contents, set the CELLPADDING attribute in the TABLE tag. All cells in a table have the same padding and spacing.

If a cell is empty, that is, the <TD> tag is immediately followed by the </TD> tag, the space occupied by the cell in the table is completely empty. That is, the cell has no content, no background color, and no border. However, suppose you have a four column table, but you have no data for the second column in one of the rows. You should still provide the second TD tag for that row, because if you leave it out the table will close the gap and move the third cell into the second column. The row will end up having three columns only, and it will not be aligned with the rest of the table

If you want an empty cell to look like other cells in the table, you can give it a period or a dash to indicate that the data is unknown, for example, <TD> - </TD>.

Syntax

```
<TD
  ALIGN="CENTER|LEFT|RIGHT"
  VALIGN="BASELINE|BOTTOM|MIDDLE|TOP"
  BGOLOR="color"
  COLSPAN="value"
  ROWSPAN="value"
  HEIGHT="pixelHeight"
  WIDTH="pixelWidth"
  NOWRAP="value"
>
...
</TD>
```

ALIGN specifies the horizontal placement of the contents of the table cell:

- CENTER centers the content within the cell.
- LEFT aligns the content with the cell's left edge (the default).
- RIGHT aligns the content with the cell's right edge.

VALIGN specifies the vertical placement of the contents of the cell:

- BASELINE aligns the content with the cell's baseline.
- BOTTOM aligns the content with the cell's bottom.

- MIDDLE centers the content within the cell (the default).
- TOP aligns the content with the cell's top.

BGCOLOR="color" sets the color of the background of the table cell.

COLSPAN="value" indicates the number of columns the cell spans. Give the number as an integer.

ROWSPAN="value" indicates the number of rows the cell spans. Give the value as an integer.

HEIGHT="pixelHeight" specifies the suggested height of the table cell, in pixels.

WIDTH="pixelWidth" specifies the suggested width of the table cell, in pixels.

NOWRAP specifies that the lines within a cell cannot be broken (that is, they do not wrap onto the next line).

Used Within TABLE and TR

Example See example 1. Simple Table.

TH - (table heading) The TH tag specifies a table cell whose contents are usually displayed in a bolder font than those of regular table cells. The intent of the TH tag is that you use it for column or row headings.

Syntax

```
<TH  
  ALIGN="CENTER|LEFT|RIGHT"  
  VALIGN="BASELINE|BOTTOM|MIDDLE|TOP"  
  BGCOLOR="color"  
  COLSPAN="value"  
  ROWSPAN="value"  
  HEIGHT="pixelHeight"  
  WIDTH="pixelWidth"  
  NOWRAP  
>  
...  
</TH>
```

ALIGN specifies the horizontal placement of the heading in the table cell:

- CENTER centers the content within the cell.
- LEFT aligns the content with the cell's left edge (the default).
- RIGHT aligns the content with the cell's right edge.

VALIGN specifies the vertical placement of the contents of the cell:

- BASELINE aligns the content with the cell's baseline.
- BOTTOM aligns the content with the cell's bottom.
- MIDDLE centers the content within the cell (the default).
- TOP aligns the content with the cell's top.

BGCOLOR="color" sets the color of the background of the table heading. This color can be overridden by a BGCOLOR tag in the TD tags within the TH tag.

COLSPAN="value" indicates the number of columns the cell spans.

ROWSPAN="value" indicates the number of rows the cell spans.

HEIGHT="pixelHeight" specifies the suggested height of the table cell, in pixels.

WIDTH="pixelWidth" specifies the suggested width of the table cell, in pixels.

NOWRAP specifies that the lines within a cell cannot be broken; that is, they do not wrap onto the next line.

Table's Attributes

border (example)

This specifies the width in pixels of the border around the table
This is in addition to the border around each cell (the "cellspacing").
The default is zero

cellspacing (example)

This gives the space in pixels between adjacent cells.
The default is usually about 3

cellpadding (example)

Specifies the space between the cell walls and contents
The default is usually about 1

width

This specifies the width of the table
In pixels (<table width="250">), or
As a percentage of the current browser window width (<table width="75%">)

rules (example) Specifies the horizontal/vertical divider lines.

Must be used in conjunction with the "border" attribute!

frame (example) Specifies which outer borders are drawn. All four are drawn if this attribute is omitted.**Table Row (tr)** Define each row in the table

Each row may contain table header (th) and table data (td) elements

Attributes:

align: Horizontal alignment

Values: "left", "center", "right", "justify", "char"

valign: Vertical alignment

Values: "top", "middle", "bottom"

Table Header (th) and Table Data (td)

Define a table cell

Attributes

Colspan: Defines a heading or cell data entry that spans multiple columns

Rowspan: Defines a heading or cell data entry that spans multiple rows

align: "left", "right", "center", "justify", "char"

e.g., the following aligns entries on a decimal point

```
<td align="char" char=".">...</td>
```

valign: "top", "bottom", "middle"

width, height: Cell width and height in pixels only (no percentages officially allowed)

HTML Frames:**Frames and Framesets**

This section discusses the tags for creating frames and frame sets. A frame is region of a window that acts as a window itself. The main window can contain multiple frames, so that different regions of the window display different contents. A frame set is a set of frames.

- **FRAME**
- **FRAMESET**
- **NOFRAMES**

FRAME - (window region)

The FRAME tag creates a frame, which is an individual, independently scrollable region of a web browser. You can think of it as a window within a window. The FRAME tag must be used within a FRAMESET tag.

The FRAMESET tag contains a set of FRAME tags, which each define a frame in the main window.

Each frame has a distinct URL that determines the content displayed by the frame.

You can specify whether or not a frame has a border, whether or not it has margins, whether or not the user can resize it dynamically, and whether or not it is scrollable.

Syntax

```
<FRAME  
  BORDERCOLOR="color"  
  FRAMEBORDER="YES"|"NO"  
  MARGINHEIGHT="marginHeight"  
  MARGINWIDTH="marginWidth"  
  NAME="frameName"  
  NORESIZE  
  SCROLLING="YES"|"NO"|"AUTO"  
  SRC="URL"  
>
```

The URL tag is required.

BORDERCOLOR="color" specifies the color of the frame's borders. Because frames share borders, Navigator must resolve any border color conflicts.

FRAMEBORDER determines whether or not the frame has visible borders.

- YES results in an outline-3D border.
- NO suppresses the 3D border.

When the FRAMEBORDER attribute appears in a FRAMESET tag, it sets a default FRAMEBORDER value for all frames in the frameset. When the FRAMEBORDER attribute appears in the FRAME tag, it applies only to that particular frame, overriding any FRAMEBORDER attribute established by an outer FRAMESET tag. A border shared between frames is plain only if all adjacent frames have the FRAMEBORDER attribute set to NO.

When neither a FRAME nor a FRAMESET tag governing that FRAME has set the FRAMEBORDER attribute, the default setting is YES.

MARGINHEIGHT="marginHeight" specifies a margin, in pixels, between the top and bottom edges of the frame and the frame contents.

MARGINWIDTH="marginWidth" specifies a margin, in pixels, between the left and right edges of the frame and the frame contents.

NAME="frameName" specifies the name of the frame. The value of the NAME attribute must begin with an alphanumeric character.

NORESIZE specifies that users cannot resize the frame. If a frame adjacent to an edge is not resizable, the entire edge is not resizable, and consequently other frames adjacent to that edge are not resizable.

SCROLLING specifies whether scrollbars are available on a frame:

- YES specifies that scrollbars are always available.
- NO specifies that scrollbars are never available.

· AUTO specifies that the browser determines whether to display scroll bars based on the size of the frame and its content. If you do not specify a value for SCROLLING, the default value is AUTO.

SRC="URL" specifies the URL for the document to be displayed in the frame. The URL cannot include an anchor name; for example FRAME SRC="doc2.html#colors" is invalid. If you do not specify the SRC attribute, the frame is displayed with no content

Used Within FRAMESET

Example See Frameset example 1. Simple Frameset with two frames.

FRAMESET - (set of frames)

The FRAMESET tag defines a set of frames that appear in a web browser window.

The FRAMESET tag contains one or more FRAME tags that each describe a frame.

The only place the FRAMESET tag can be used is in a frame definition document, which is an HTML document that contains the FRAMESET and FRAME tags that describe the framesets and frames that make up a Navigator window.

An HTML document that contains a FRAMESET tag cannot contain a BODY tag.

A frameset can specify that its frames are laid out in rows or columns. If you want your frameset to have rows and columns, rather than just rows or columns, you can use FRAMESET tags nested inside FRAMESET tags.

For example, you could define a frameset that has two columns, where the first column contains a frameset that has two rows and the second column contains a frameset that has 4 rows.

You can specify the border thickness for all frames in a top-level frameset. You can also specify whether or not all frames in a set display their border by default and what color the border uses. Individual frames in the set can override the default to specify whether or not they display their border, and what color their border uses.

You can specify actions to occur when the window displaying the frameset gets or loses focus, and you can specify actions to occur when the frameset is loaded or unloaded.

When you define a link, (using the <A HREF> tag) you can specify in which frame the destination document is displayed, by giving the name of the frame as the value of the link's TARGET attribute. It is possible to define a link to change the content of multiple frames in one go by using the ONCLICK attribute of the link. To do this, set the value of the link's ONCLICK attribute to JavaScript code that changes the location (source) of one or more frames.

For example, the file frameset.htm opens a web page that has three frames. If you click a link in the left frame, both the other frames update. You can open the file framtoc.htm and view its source to see the code for the links that update multiple frames.

The NOFRAMES tag is used inside a FRAMESET tag to provide alternative content for browsers that cannot display frames.

Syntax

```
<FRAMESET  
COLS="columnWidthList"  
ROWS="rowHeightList"  
BORDER="pixWidth"  
BORDERCOLOR="color"  
FRAMEBORDER="YES"|"NO"  
ONBLUR="JScode"  
ONFOCUS="JScode"  
ONLOAD="JScode"
```

```
ONUNLOAD="JScode"  
>  
...  
</FRAMESET>
```

You must supply at least one of the COLS or ROWS attributes.

COLS="columnWidthList" specifies a comma-separated list of values giving the width of each frame in the frameset. If one of the values is missing, the browser sizes the corresponding frame to fit the space available.

The browser may approximate some values to make the total width of the columns equal to the width of the window.

The value of each item in the columnWidthList can be one of the following:

- Width of a frame in pixels.
- Width of a frame as a percentage of its parent frame or window.
- An asterisk (*), which means "as much space as possible," which is the space left over after space has been allocated to all columns that specify their width as pixel value or a percentage value . The total available left-over space is divided equally between all columns that use an asterisk.

ROWS="rowHeightList" specifies a comma-separated list of values giving the height of each frame in the frameset. If one of the values is missing, the corresponding frame is sized to fit the space available. The browser may approximate some values to make the total height of the rows equal to the height of the window. Each item in rowHeightList can be one of the following:

- Height of a frame in pixels.
- Height of a frame as a percentage of the parent frame or window.
- An asterisk (*) which means "as much space as possible," which is the space left over after space has been allocated to all rows that specify their height as pixel value or a percentage value . The total available left-over space is divided equally between all rows that use an asterisk.

BORDER="pixWidth" specifies the thickness of frame borders for all frames in an outermost frameset. A setting of 0 causes all frames in the frameset to have no border between them. A setting of 3 causes a border of 3 pixels. If no BORDER tag is present, the default is 5 pixels. The BORDER tag can be used only on an outermost FRAMESET tag.

BORDERCOLOR="color" specifies the color of a frame's borders. Because frame borders are shared, Navigator must resolve any border color conflicts.

1. Any BORDERCOLOR attribute appearing in the outermost FRAMESET has the lowest priority.
2. This attribute is overridden by any attribute used in a nested FRAMESET tag.
3. Finally, any BORDERCOLOR attribute in the current FRAME tag overrides all previous FRAMESET tag settings. If there is a conflict for two colors of equal priority both set on the same edge, the behavior is undefined.

FRAMEBORDER determines how frame borders are displayed.

- YES results in an outline-3D border.
- NO suppresses the 3D border.

When the FRAMEBORDER attribute appears in the FRAMESET tag, it sets a default FRAMEBORDER value for all frames in that frameset. When the FRAMEBORDER attribute appears in a FRAME tag, it applies only to that particular frame, overriding any FRAMEBORDER attribute specified

by an outer FRAMESET tag. A border shared between frames is plain only if all adjacent frames have the FRAMEBORDER attribute set to NO. When neither a FRAME nor a FRAMESET tag governing that FRAME has set the FRAMEBORDER attribute, the default setting is YES.

ONBLUR="JScript" specifies JavaScript code to execute when the window containing the frameset loses focus.

ONFOCUS="JScript" specifies JavaScript code to execute when the window containing the frameset gets focus.

ONLOAD="JScript" specifies JavaScript code to execute when the frameset is loaded into the frame.

ONUNLOAD="JScript" specifies JavaScript code to execute when the frameset is unloaded (exited).

Frameset Example 1. Simple Frameset With Two Frames

The following example creates a set of two frames. The frameset is kept in a document file, such as index.html, that contains no other information. When users open this page in their web browser, the FRAMESET tag loads the individual URLs referenced in the FRAME tags.

```
<HTML>
<HEAD> <TITLE>Simple Frame Set Example</TITLE></HEAD>
<FRAMESET COLS="20%,80%" BORDER=10>
<FRAME SRC="simpltoc.htm" NAME="exampletoc">
<FRAME SRC="forms.htm" NAME="examplecontent">
<NOFRAMES>You must use a browser that can display frames
to see this page. </NOFRAMES>
</FRAMESET>
</HTML>
```

The two frames appear as columns because COLS is specified within the FRAMESET tag. The left frame uses 30% of the available space, and the right frame uses the remaining 70% of the space. By default, the frames in this example have scrollbars and are resizable, because no values are specified for the SCROLLING and NORESIZE attributes.

Example 2: Nested Frames

The following example creates a frameset that contains a nested frameset. The outermost frameset has two columns. The second column contains a nested frameset that has two rows.

```
<HTML><HEAD><TITLE>Frame Set Example</TITLE></HEAD>
<FRAMESET COLS="20%,*">
<NOFRAME>You must use a browser that can display frames
to see this page.</NOFRAME>
<FRAME SRC="frametoc.htm" NAME="noname">
<FRAMESET ROWS="30%,*">
<FRAME SRC="frtoc1.htm" NAME="toptoc">
<FRAME SRC="frstart.htm" NAME="outer">
</FRAMESET>
</FRAMESET>
</HTML>
```

NOFRAMES - (alternative text for frames)

The NOFRAMES tag specifies content that is displayed by browsers that do not know how to display frames. Browsers that can display frames ignore all text in the NOFRAMES tags unless a file called by a frame tag is missing or unreachable by the browser.

Place the NOFRAMES tag within the FRAMESET tag.

Syntax

<NOFRAMES>...</NOFRAMES>

Used Within <FRAMESET>

HTML Forms:

- Form is a layout component used in web page to interact the user.
- Form is also used to link the another page or another form based on the action .

<INPUT TYPE="text" NAME="name" VALUE ="value" SIZE=size>	Creates Text Field
<INPUT TYPE="password" NAME="name" VALUE="value" SIZE=size>	Password Field
<INPUT TYPE="hidden" NAME="name" VALUE="value">	Hidden Field
<INPUT TYPE="checkbox" NAME="name" VALUE="value">	Checkbox
<INPUT TYPE="radio" NAME="name" VALUE="value">	Radio Button
<SELECT NAME="name" SIZE=1> <OPTION SELECTED>one <OPTION>Two : </SELECT>	Dropdown List
<SELECT NAME="name" SIZE=n MULTIPLE>	Scrolling List
<TEXTAREA ROWS=yy COLS=xx NAME="name" .. </TEXTAREA>	Multiple Test Fields
<INPUT TYPE="submit" VALUE="Submit" >	Submit button
<INPUT TYPE="reset" VALUE="Reset" >	Reset Button

- List of component

Text
Text Area
Label
Button
Check box

Radio button

Menus

1. Text

- Text component is used to insert a text in web page.
- <Input> tag is used to insert a text component in document.
- General form

```
<input type="text" size=25 value= " ">
```

2. Text Area

- Text Area component is used to feed multiple line of text.
- Attribute in text area component.
- Row: Row attribute used to denote the total number of rows in the text area.
- Column: Column attribute is used to denote the number of column in the text area.
- Name: Name attribute is used to denote the name of the text area.
- Wrap: Wrap attribute is used to wrap the text inside the text area.
- <text area> tag is used to insert the text area in the web document.
- General form:

```
<text area cols="value" rows="values" > ---</text area>
```

- Example:

```
<text area cols="50" rows="40" name="name"> --</text area>
```

3. Label

- Label tag is used to create text box to fill the text.
- This tag has starting tag (<label>) and ending tag (</label>).
- Example

```
<label> Name <input type="text" size=40>--</label>
```

4. Check box

- Check box component is used to place the checkbox in the web document.
- General form:

```
<input type="checkbox"> ----- </input>
```

MENU - (list of simple items)

The MENU tag displays a list of simple items. This tag works just like the UL tag.

Use the LI tag to designate the individual menu items.

This tag is being deprecated because it has become obsolete in favor of the UL tag.

Syntax

```
<MENU>
```

```
...
```

```
</MENU>
```

Example

The following example creates a list of three short items:

```
<P>HTML Menu Example:</P>
```

```
<MENU>
```

```
<LI> Windows 9x
```

```
<LI> Windows NT
```

```
<LI> Windows 2000
```

```
</MENU>
```

Forms

This section discusses the tags for creating forms.

- ☐ **FORM**
- ☐ **INPUT**

INPUT TYPE="BUTTON"
INPUT TYPE="CHECKBOX"
INPUT TYPE="FILE"
INPUT TYPE="HIDDEN"
INPUT TYPE="IMAGE"
INPUT TYPE="PASSWORD"
INPUT TYPE="RADIO"
INPUT TYPE="RESET"
INPUT TYPE="SUBMIT"
INPUT TYPE="TEXT"

- ☐ **SELECT**
- ☐ **OPTION**
- ☐ **ISINDEX**

FORM - (form for user input)

The FORM tag creates an HTML form. The form can contain interface elements such as text fields, buttons, checkboxes, radio buttons, and selection lists that let users enter text and make choices. Each interface element in the form must be defined with an appropriate tag, such as <INPUT> or <SELECTION>. All elements in the form must be defined between the <FORM> and </FORM> tags. As well as user input elements, the form can contain other elements such as headings, paragraphs, tables, and so on.

When the form is displayed in a web browser, the user can fill it out by making choices and entering text using the interface elements, and then submit the form by clicking a "Submit" button.

Kinds of Interface Elements

Several kinds of form elements can be defined using the INPUT tag, which uses the TYPE attribute to indicate the type of element, such as button, checkbox, and so on.

Two other kinds of interface elements you can put in a form are selection lists and text areas. Selection lists act as menus and are defined with the SELECT tag. Multi-line text-entry fields are defined with the TEXTAREA tag.

Submit Buttons and CGI Programs

To enable the form to process the data that the user enters, it must have a "Submit" button, which is a button defined by an <INPUT TYPE="SUBMIT"> or an <INPUT TYPE="IMAGE"> tag.

The action invoked when the user clicks a "Submit" button is defined by the ACTION attribute of the FORM tag. The value of the ACTION attribute is usually a URL that points to a CGI program. A CGI program runs on a server, processes arguments sent by the form, and returns data to the browser.

The value of the form's METHOD attribute also affects the way the CGI program is invoked. It is beyond the scope of this reference to provide details of CGI programming, but many fine books are available on the subject, and also lots of information is available on the web.

ONCLICK and ONSUBMIT

You can also define OnClick event handlers for several kinds of input elements. An OnClick event handler is a piece of JavaScript code that is executed when the element is clicked.

The FORM tag has an optional ONSUBMIT attribute, whose value is a JavaScript event handler that executes when a "Submit" button in the form is pressed. If the JavaScript code returns false, the form's action ends there, and the URL specified by the ACTION attribute is not invoked.

If the JavaScript code returns anything else, the URL specified by the ACTION attribute is invoked. For example, you could use the ONSUBMIT attribute to check whether or not the user really wants to submit the form.

Name/Value Pairs

When a form is submitted, the data contained in the form is sent to the invoked CGI program as a series of name/value pairs. The name portion of each pair is the name of an interface element as specified by its NAME attribute. In most cases the value portion is the value displayed by the element, for example, the text displayed in a text field.

Nesting Forms

A document can have multiple forms, but forms cannot be nested -- you cannot have a form within a form. If your document uses positioned HTML content, each form must be completely contained within one positioned block.

Syntax

```
<FORM  
ACTION="serverURL"  
ENCTYPE="encodingType"  
METHOD="GET"|"POST"  
NAME="formName"  
ONRESET="JScode"  
ONSUBMIT="JScode"  
TARGET="windowName"  
>  
  
...  
</FORM>
```

The ACTION attribute is required if any action is to occur when the user presses a "Submit" button in the form.

ACTION="serverURL" specifies the URL of the program to be invoked when the form is submitted. The action can also be a mailto: URL if the form results are to be mailed to someone.

ENCTYPE="encodingType" specifies the MIME encoding of the data sent:

"application/x-www-form-urlencoded" (the default), is usually used if the METHOD attribute has the value POST.

"multipart/form-data" is used when the form contains a file upload element (INPUT TYPE="FILE").

METHOD specifies how information is sent to program invoked by submitting the form.

GET (the default) appends the input information to the URL which on most receiving systems becomes the value of the environment variable QUERY_STRING.

POST sends the input information in a data body that is available on stdin with the data length set in the environment variable CONTENT_LENGTH.

NAME="formName" specifies the name of the form. The name is not displayed on the form.

JavaScript can use the NAME attribute to differentiate different forms if there are multiple forms on a page.

ONRESET="JScode" specifies JavaScript code that executes when a user resets the form by using a RESET button.

ONSUBMIT="JScode" specifies JavaScript code that executes when a user submits the form by clicking a "Submit" button. You can use the ONSUBMIT attribute to prevent a form from being submitted; to do so, put a return statement that returns false in the JavaScript code. Any other returned value lets the form submit. If you omit the return statement, the form is submitted.

TARGET="windowName" specifies the window that displays the data returned by the invoked program.

Example

The following example creates a form called LoginForm that contains text fields for user name and password, a submit button, and a cancel button.

```
<FORM NAME="LoginForm" METHOD=POST ACTION="urltoInvoke">
<P>User name:
<INPUT TYPE="text" NAME="userName" SIZE="10">
<P>Password:
<INPUT TYPE="password" NAME="password" SIZE="12">
<P><INPUT TYPE="submit" VALUE="Log in">
<INPUT TYPE="button" VALUE="Cancel" onClick="window.close()">
</FORM>
```

INPUT - (input element in a form)

- The INPUT tag defines a form element that can receive user input. The TYPE attribute determines the specific sort of form element to be created. TYPE can be one of the following:
- BUTTON places a button on an HTML form. Use JavaScript code to make the button perform an action you define.
- CHECKBOX places a toggle switch on an HTML form, letting the user set a value on or off.
- FILE places an element on an HTML form letting the user supply a file as input. When the form is submitted, the content of the specified file is sent to the server along with the other form data.
- HIDDEN specifies an invisible text element. A hidden element is used for passing information to the server when a form is submitted.
- IMAGE places an image, serving as a custom button, on an HTML form. When a user clicks an image element, the form is submitted to the server.
- PASSWORD places a text input field on an HTML form. Each character typed in the field is displayed as a character such as * or a black dot to conceal the actual value.
- RADIO places a radio button on an HTML form. Radio buttons can be grouped into sets, and only one button per set can be selected at a time.
- RESET places a reset button on an HTML form. When a user clicks a reset button, all elements in the form are reset to their default values.
- SUBMIT places a submit button on an HTML form. When a user presses a submit button, the form is submitted.
- TEXT places a single line text input field on an HTML form. A text field lets the user enter text.

INPUT TYPE="BUTTON"

A button appears in the form. You must specify JavaScript code as the value of the ONCLICK attribute to determine what happens when the user clicks the

button.

Syntax

```
<INPUT TYPE="BUTTON"
NAME="buttonName"
VALUE="buttonText"
ONCLICK="JScode"
>
```

NAME="buttonName" specifies the name of the button. The name does not appear in the form.

VALUE="buttonText" specifies the text to be displayed in the button.

ONCLICK="JScode" specifies JavaScript code to execute when a user clicks the button.

Example

```
<FORM METHOD=POST ACTION="/cgi-bin/example.cgi">
<INPUT TYPE="button" VALUE="Close Window"
onClick="window.close();">
</FORM>
```

INPUT TYPE="CHECKBOX" A checkbox is a toggle that the user can select (switch on) or deselect (switch off.)

Syntax

```
<INPUT TYPE="CHECKBOX"
CHECKED
NAME="name"
ONCLICK="JScode"
VALUE="checkboxValue"
>
```

CHECKED indicates that the checkbox is displayed with a tick mark to indicate that it is selected.

NAME="name" specifies the name of the input element. This value is the name portion of the name/value pair for this element that is sent to the server when the form is submitted. The name is not displayed on the form.

ONCLICK="JScode" specifies JavaScript code to execute when a user clicks the checkbox.

VALUE="checkboxValue" specifies the value to be returned to the server if the checkbox is selected when the form is submitted. The default value is ON, but you can specify a different value if you want. When the form is submitted, only the name/value pairs for selected checkboxes are sent to the invoked CGI program.

Example

```
<P>Specify your music preferences (check all that apply):</P>
<BR><INPUT TYPE="checkbox" NAME="musicpref_rnb" CHECKED>
R&B
<BR><INPUT TYPE="checkbox" NAME="musicpref_jazz" CHECKED>
Jazz
<BR><INPUT TYPE="checkbox" NAME="musicpref_blues" CHECKED>
Blues
```

```
<BR><INPUT TYPE="checkbox" NAME="musicpref_newage" CHECKED>  
New Age
```

INPUT TYPE="FILE" This places an element on an HTML form that lets the user supply a file as input. When the form is submitted, the content of the specified file is sent to the server as the value portion of the name/value pair for this input element.

If a form contains a file input element, the value of the ENCTYPE attribute of the FORM tag should be "multipart/form-data".

Syntax

```
<INPUT TYPE="FILE"  
NAME="name"  
VALUE="filename"  
>
```

NAME=name specifies the name of the input element. This value is used as the name portion of the name/value pair for this element that is sent to the server when the form is submitted. The name is not displayed on the form.

VALUE=filename specifies the initial value of the input element.

Example

```
<FORM ENCTYPE="multipart/form-data"  
ACTION="/cgi-bin/example.cgi" METHOD="POST">  
<P>File name:  
<INPUT TYPE="file">  
</FORM>
```

INPUT TYPE="HIDDEN"

A hidden input element is an invisible element whose main purpose is to contain data that the user does not enter. This data gets sent to the invoked CGI program when the form is submitted.

This tag provides a mechanism for delivering a value to the CGI program without the user having entered it, but note that it is not very hidden because the user can discover it by viewing the document source.

Syntax

```
<INPUT TYPE="HIDDEN"  
NAME="name"  
VALUE="value"  
>
```

NAME="name" specifies the name of the input element. This value is the name portion of the name/value pair sent to the invoked CGI program when the form is submitted. The name is not displayed on the form.

VALUE="value" specifies the initial value of the input element.

Example

This example creates a form with a hidden element, DefaultPass, that stores the initial value of the password field.

```
<FORM NAME="LoginForm" METHOD=POST ACTION="/cgibin/  
example.cgi">  
<P>Password:  
<INPUT TYPE="password" NAME="password" SIZE="12"  
VALUE="treasure">
```

```
<INPUT TYPE="hidden" NAME="DefaultPass" VALUE="treasure">
</FORM>
```

INPUT TYPE="IMAGE"

This places an image, serving as a custom button, on an HTML form. When a user clicks the image, the form is submitted to the server.

Syntax

```
<INPUT TYPE="IMAGE"
ALIGN="LEFT"|"RIGHT"|"TOP"|"ABSMIDDLE"|"ABSBOTTOM"|
"TEXTTOP"|"MIDDLE"|"BASELINE"|"BOTTOM"
NAME="name"
SRC="location"
>
```

ALIGN specifies the alignment of the image in relation to the surrounding text. If you do not specify a value for ALIGN, Navigator uses BOTTOM as the default. The possible values are LEFT, RIGHT, TOP, ABSMIDDLE, ABSBOTTOM, TEXTTOP, MIDDLE, BASELINE, and BOTTOM. See the section "IMG" for a description of the values.

NAME=name specifies the name of the input element. This value is used as the name portion of the name/value pair for this element that is sent to the invoked CGI program when the form is submitted. The name is not displayed on the form. When Navigator sends the offsets of the image to the server, it sends them as name.x and name.y.

SRC="location" specifies the URL of the image to be displayed in the document.

Example

```
<CENTER><INPUT TYPE="image" SRC="signnow.gif"></CENTER>
```

INPUT TYPE="PASSWORD" A password element is a text input field in which each character typed is displayed as a character such as * or a black dot to conceal the actual value.

Syntax

```
<INPUT TYPE="PASSWORD"
MAXLENGTH="maxChar"
NAME="name"
ONSELECT="JScript"
SIZE="charLength"
VALUE="textValue"
>
```

MAXLENGTH="maxChar" specifies the maximum number of characters a password box can accept.

NAME="name" specifies the name of the input element. This value is used as the name portion of the name/value pair for this element that is sent to the server when the form is submitted. The name is not displayed on the form.

ONSELECT="JScript" specifies JavaScript code to execute when a user selects some of the text in the text element.

SIZE="charLength" specifies the length of the input field, in characters. The value should be an integer.

VALUE="textValue" specifies the initial value of the password, if any.

Example

```
<P>Password:
<INPUT TYPE="password" NAME="password" VALUE="" SIZE="25">
```

INPUT TYPE="RADIO" A radio element is a radio button. A set of radio buttons consists of multiple radio buttons that all have the same NAME attribute. Only one radio button in the set can be selected at one time. When the user selects a button in the set, all other buttons in the set are deselected. If one radio button in a set has the CHECKED attribute, that one is selected when the set is first laid out on the window.

Syntax

```
<INPUT TYPE="RADIO"  
CHECKED  
NAME="name"  
ONCLICK="JScode"  
VALUE="buttonValue"  
>
```

CHECKED indicates that the radio button is selected.

NAME="name" specifies the name of the input element. This value is used as the name portion of the name/value pair for this element that is sent to the invoked CGI program when the form is submitted. The name is not displayed on the form. All radio buttons that have the same name constitute a radio group; only one radio button of a group can be set at one time.

ONCLICK="JScode" specifies JavaScript code to execute when a user clicks the radio button.

VALUE="value" specifies the value that is returned to the server when the radio button is selected and the form is submitted. Only name/value pairs for radio buttons that are selected are sent to the invoked CGI program. The value defaults to ON.

Example

The following example creates a radio button group.

```
<P>Category:  
<BR><INPUT TYPE="radio" NAME="category" VALUE="liv" CHECKED>  
Living  
<BR><INPUT TYPE="radio" NAME="category" VALUE="din"> Dining  
<BR><INPUT TYPE="radio" NAME="category" VALUE="bed"> Bedroom
```

INPUT TYPE="RESET" When a user presses a reset button, all elements in the form are reset to their default values

Syntax

```
<INPUT TYPE="RESET"  
NAME="name"  
ONCLICK="JScode"  
VALUE="label"  
>
```

NAME="name" specifies the name of the input element.

ONCLICK="JScode" specifies JavaScript code to execute when a user clicks the button.

VALUE="label" specifies the text to display on the face of the reset button.

Example

This example displays a text element with the default value CA and a reset button labelled Clear Form. If the user types a state abbreviation in the text element and then clicks the Clear Form button, the original value of CA is restored.

```
<FORM>  
<P>State: <INPUT TYPE="text" NAME="state" VALUE="CA" SIZE="2">
```

```
<P><INPUT TYPE="reset" VALUE="Clear Form">
</FORM>
```

INPUT TYPE="SUBMIT" When a user clicks a submit button, the form is submitted, which means that the ACTION specified for the form is invoked.

Syntax

```
<INPUT TYPE="SUBMIT"
NAME="name"
VALUE="label">
```

NAME="name" specifies the name of the input element. The name is not displayed on the form.

VALUE="label" specifies the text to display on the face of the submit button.

Example

```
<INPUT TYPE="submit" NAME="SubmitButton" VALUE="Done">
```

INPUT TYPE="TEXT" A text element is a single line text input field in which the user can enter text.

Syntax

```
<INPUT TYPE="TEXT"
MAXLENGTH="maxChars"
NAME="name"
ONBLUR="Scode"
ONCHANGE="JScode"
ONFOCUS="Scode"
ONSELECT="JScode"
SIZE="lengthChars"
VALUE="text"
>
```

MAXLENGTH="maxChars" specifies the maximum number of characters a text box can accept.

NAME="name" specifies the name of the input element. This value is used as the name portion of the name/value pair for this element that is sent to the invoked CGI program when the form is submitted. The name is not displayed on the form.

ONBLUR="JScode" specifies JavaScript code to execute when the text element loses keyboard focus.

ONCHANGE="JScode" specifies JavaScript code to execute when the text element loses focus and its value has been modified.

ONFOCUS="JScode" specifies JavaScript code to execute when a user clicks the text element. See the JavaScript Guide for more information.

ONSELECT="JScode" specifies JavaScript code to execute when a user selects some of the text in the text element.

SIZE="lengthChars" specifies the length of the input field, in characters.

VALUE="text" specifies the initial value of the text element.

Example

```
<P>Last name:
<INPUT TYPE="text" NAME="last_name" VALUE="" SIZE="25">
```

SELECT - (selection list in a form)

The SELECT tag defines a selection list on an HTML form. A selection list displays a list of options from which the user can select an item. If the MULTIPLE attribute is supplied, users can select

multiple options from the list at a time. If the **MULTIPLE** attribute is not supplied users can select only one option in the list at a time.

The **SIZE** attribute specifies how many options in the list are displayed at one time. For multiple-selection lists, if you do not specify the **SIZE** attribute, the browser displays some, maybe all, of the options. For single-selection lists, by default Navigator displays the list as a drop-down menu that initially shows only one option. The user can click the list to display the rest of the options. If you specify the **SIZE** attribute, the list is displayed as a scrolling list that shows the specified number of options, regardless of whether the list allows single or multiple selection..

The **SELECT** tag should be used between **<FORM>** and **</FORM>** tags. Use the **OPTION** tag to define options in the list. When the form containing the selection list is submitted to the server, a name/value pair is sent for each selected option in the list.

Syntax

```
<SELECT  
  NAME="selectName"  
  MULTIPLE  
  ONBLUR="JScode"  
  ONCHANGE="JScode"  
  ONCLICK="JScode"  
  ONFOCUS="fScode"  
  SIZE="listLength"  
>  
<OPTION...>  
...  
<OPTION ...>  
</SELECT>
```

MULTIPLE specifies that multiple items can be selected. If this attribute is omitted, only one item can be selected from the list. If multiple selection is enabled, the user usually needs to hold down the Shift key to select additional items.

NAME="selectName" specifies the name of the select element. This value is the name portion of the name/value pair sent to the invoked CGI program when the form is submitted. The name is not displayed on the form.

ONBLUR="blurJScode" specifies JavaScript code to execute when the select element loses focus.

ONCHANGE="changeJScode" specifies JavaScript code to execute when the select element loses focus and its value has been modified.

ONCLICK="JScode" specifies JavaScript code to execute when a user clicks an item in the list.

ONFOCUS="focusJScode" specifies JavaScript code to execute when the element gets focus.

SIZE="ListLength" specifies the number of options visible when the form is displayed. If the list contains more options than specified by size, the list is displayed with scrollbars.

Used Within FORM

Select Example 1: Single Item Selection

```
<FORM>  
<B>Shipping method:</B><BR>  
<SELECT>  
<OPTION> Standard
```

```
<OPTION SELECTED> 2-day
<OPTION> Overnight
</SELECT>
</FORM>
```

Example 2: Multiple Selection

```
<FORM>
...
<B>Music types for your free CDs:</B><BR>
<SELECT NAME="music_type_multi" MULTIPLE>
<OPTION> R&B
<OPTION> Jazz
<OPTION> Blues
<OPTION> Reggae
</SELECT>
</FORM>
```

Example 3: Multiple Selection With Default

In the following example, all seven options can be chosen, but bananas are selected by default. The list is displayed as a scrollable menu that fits four options at a time.

```
<FORM>
<SELECT NAME="fruit_choices" MULTIPLE>
<OPTION>Apples
<OPTION SELECTED>Bananas
<OPTION>Cherries
<OPTION>Oranges
<OPTION>Pineapple
<OPTION>Figs
<OPTION>Guava
</SELECT>
</FORM>
```

OPTION - (option in a SELECT list) The OPTION tag specifies an option in a selection list. Use the OPTION tag inside a SELECTION tag. When the form containing the selection list is submitted to the server, a name/value pair is sent for each selected option in the list. The value portion of an option is the value of the VALUE attribute, if it has one, otherwise, it is the text that follows the <OPTION> tag.

Syntax

```
<OPTION
VALUE="optionValue"
SELECTED
>
...
</OPTION>
```

VALUE="OptionValue" specifies a value that is returned to the server when the option is selected and the form is submitted. When no VALUE attribute is present, the value returned is the same as the text following the <OPTION> tag.

SELECTED specifies that the option is selected by default.

Used within FOR

Example See example 1:Single Item Selection.

ISINDEX - (searchable index)

The ISINDEX tag causes the web page to display a text entry field in which the user can type a string. The intent of this tag is that it "switches on searching" in the page, but in reality, this tag is useful only if the page is generated by a CGI program.

The intent is that when the user enters text into the text entry field and presses the Return key (or clicks an appropriate button on the browser), the CGI program is invoked again, with the arguments generated from the data in the text field. The browser outputs a new page whose content is determined by what the user entered in the text field.

The CGI program should test for the presence of arguments. If there are none, it should display a default page that includes the ISINDEX tag in the header. If there are arguments, the script does whatever it needs to do. The string entered by the user is the first argument, and the language your script uses determines how you access the first argument.

It is beyond the scope of this reference to provide details on CGI programming, but many fine books are available on the subject, and lots of information is available on the web.

Note that ISINDEX does not require a closing tag.

Syntax

```
<ISINDEX PROMPT="text" >
```

PROMPT="text" specifies the text that appears as the search prompt in the browser.

Used Within HEAD

Example

The following snippet of code from a CGI program generates the header for an HTML page. When the page is displayed, it has a text entry field whose prompt is "Enter a search keyword:".

```
cat << EOF
<HEAD>
<ISINDEX PROMPT="Enter a search keyword:">
</HEAD>
```

UNIT - II

Client Side Programming : JavaScript

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)

What can a JavaScript do?

- **JavaScript can put dynamic text into an HTML page**
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

The Real Name is ECMAScript. JavaScript is an implementation of the ECMAScript language standard. ECMAScript is developed and maintained by the ECMA organization. ECMA-262 is the official JavaScript standard. The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996. The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997. The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998. The development of the standard is still in progress.

Structure of Java Script

```
<html>
<head>
<script type="text/javascript">
    document.writeln("Helo World!");
    -----
</script>
</head>
<body>
    ---
    ---
</body>
</html>
```

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.

So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends. It is possible to have script code inside the body tag also as shown below. If it is placed inside the body tag, the script will be executed when the content of HTML document is displayed.

```
<html>
```

```
<body>
<script type="text/javascript">
.....
</script>
</body>
</html>
```

The **document.write** command is a standard JavaScript command for writing output to a page. By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, are placed in functions. It is a good practice to put all your functions in the head section, this way they are all in one place and do not interfere with page content.

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, or at a later event, such as when a user clicks a button. When this is the case we put the script inside a function

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, and you can have scripts in both the body and the head section at the same time.

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>
```

```
<body onload="message()">
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>

</html>
```

Using an External JavaScript

JavaScript can also be placed in external files. External JavaScript files often contains code to be used on several different web pages. External JavaScript files have the file extension .js. External script cannot contain the <script></script> tags. To use an external script, point to the .js file in the "src" attribute of the <script> tag:

Example

```
<html>
<head>
<script type="text/javascript" src="xxx.js"></script>
</head>
<body>
</body>
</html>
```

JavaScript Variables

JavaScript variables are used to hold values or expressions. A variable can have a short name, like x, or a more descriptive name, like carname. Rules for JavaScript variable names:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter or the underscore character

Because JavaScript is case-sensitive, variable names are case-sensitive.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables. You can declare JavaScript variables with the **var** keyword:

```
var x;
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet). However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Volvo";
```

After the execution of the statements above, the variable **x** will hold the value 5, and **carname** will hold the value **Volvo**.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared. These statements:

```
x=5;  
carname="Volvo";
```

have the same effect as:

```
var x=5;  
var carname="Volvo";
```

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;  
var x;
```

After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

The Lifetime of JavaScript Variables

If you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that **y=5**, the table below explains the arithmetic operators:

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

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables. Given that **x=10** and **y=5**, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
=	x=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Comparison Operators Comparison operators are used in logical statements to determine equality or difference between variables or values. Given that x=5, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Logical Operators

Logical operators are used to determine the logic between variables or values. Given that **x=6 and y=3**, the table below explains the logical operators:

Operator	Description	Example
&&	And	(x < 10 && y > 1) is true
	Or	(x==5 y==5) is false
!	Not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename=(condition)?value1:value2

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true

- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if... else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
{
  code to be executed if condition is true
}
```

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
{
  document.write("<b>Good morning</b>");
}
</script>
```

If...else Statement

Use the if ... else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
{
  code to be executed if condition is true
}
else
{
  code to be executed if condition is not true
}
```

Example

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.
```

```
var d = new Date();
var time = d.getHours();

if (time < 10)
{
    document.write("Good morning!");
}
else
{
    document.write("Good day!");
}
</script>
```

If...else if...else Statement

Use the if...else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
{
    code to be executed if condition1 is true
}
else if (condition2)
{
    code to be executed if condition2 is true
}
else
{
    code to be executed if condition1 and condition2 are not true
}
```

Example

```
script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
    document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
    document.write("<b>Good day</b>");
}
else
{
    document.write("<b>Hello World!</b>");
}
</script>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
    execute code block 1
    break;
case 2:
    execute code block 2
    break;
default:
    code to be executed if n is different from case 1 and 2
}
```

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
    alert("I am an alert box!");
}
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Show alert box" />
</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
{
    alert("You pressed OK!");
}
else
{
    alert("You pressed Cancel!");
}
}
</script>
</head>
<body>
<input type="button" onclick="show_confirm()" value="Show confirm box" />
</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
{
    document.write("Hello " + name + "! How are you today?");
}
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript Functions

A function will be executed by an event or by a call to the function.

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function. A function contains code that will be executed by an event or by a call to the function. You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file). Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function functionname(var1,var2,...,varX)
{
  some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function. The word *function* must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name

JavaScript Function Example

Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
  alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```

The return Statement

The return statement is used to specify the value that is returned from the function. So, functions that are going to return a value must use the return statement. The example below returns the product of two numbers (a and b):

Example

```
<html>
```

```
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>

</body>
</html>
```

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (variable=startvalue;variable<=endvalue;variable=variable+increment)
{
code to be executed
}
```

Example

The example below defines a loop that starts with `i=0`. The loop will continue to run as long as `i` is less than, or equal to 5. `i` will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the `<=` could be any comparing statement.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
```

```
document.write("The number is " + i);  
document.write("<br />");  
}  
</script>  
</body>  
</html>
```

JavaScript While Loop

Loops execute a block of code a specified number of times, or while a specified condition is true.

The while Loop

The while loop loops through a block of code while a specified condition is true.

Syntax

```
while (variable<=endvalue)  
{  
  code to be executed  
}
```

Note: The <= could be any comparing operator.

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

Example

```
<html>  
<body>  
<script type="text/javascript">  
var i=0;  
while (i<=5)  
{  
  document.write("The number is " + i);  
  document.write("<br />");  
  i++;  
}  
</script>  
</body>  
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code **ONCE**, and then it will repeat the loop as long as the specified condition is true.

Syntax

do
{

```
    code to be executed
}
while (variable<=endvalue);
```

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
    document.write("The number is " + i);
    document.write("<br />");
    i++;
}
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
    if (i==3)
    {
        break;
    }
    document.write("The number is " + i);
    document.write("<br />");
}
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    continue;
  }
  document.write("The number is " + i);
  document.write("<br />");
}
</script>
</body>
</html>
```

JavaScript For...In Statement

The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

```
for (variable in object)
{
  code to be executed
}
```

Note: The code in the body of the for...in loop is executed once for each element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

Example

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";

for (x in mycars)
{
  document.write(mycars[x] + "<br />");
}
```



```
}  
</script>  
  
</body>  
</html>
```

JavaScript Events

Events are actions that can be detected by JavaScript.

Events

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript. Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input field in an HTML form
- Submitting an HTML form
- A keystroke

Events are normally used in combination with functions, and the function will not be executed before the event occurs.

Event Association

Events are associated with HTML tags. The definitions of the events described below are as follows:

WEB TECHNOLOGY

Event handler	Applies to:	Triggered when:
onAbort	Image	The loading of the image is cancelled.
onBlur	Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).
onChange	Select, Text, TextArea	The data in the form element is changed by the user.
onClick	Button, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
onDbClick	Document, Link	The object is double-clicked on.
onError	Image	A JavaScript error occurs.
onFocus	Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, TextArea	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
onKeyDown	Image, Link, TextArea	The user presses a key.
onKeyPress	Image, Link, TextArea	The user presses or holds down a key.
onKeyUp	Image, Link, TextArea	The user releases a key.
onLoad	Image, Window	The whole page has finished loading.
onMouseDown	Button, Link	The user presses a mouse button.
onMouseMove	None	The user moves the mouse.
onMouseOut	Image, Link	The user moves the mouse away from the object.
onMouseOver	Image, Link	The user moves the mouse over the object.
onMouseUp	Button, Link	The user releases a mouse button.
onMove	Window	The user moves the browser window or frame.
onReset	Form	The user clicks the form's Reset button.
onResize	Window	The user resizes the browser window or frame.
onSelect	Text, Textarea	The user selects text within the field.
onSubmit	Form	The user clicks the form's Submit button.
onUnload	Window	The user leaves the page.

JavaScript Objects

Object oriented Programming is an important aspect of JavaScript. It is possible to use built-in objects available in JavaScript. It is also possible for a JavaScript programmer to define his own objects and variable types. In this JavaScript tutorial, you will learn how to make use of built-in objects available in JavaScript.

Built-in objects in JavaScript:

Some of the built-in objects available in JavaScript are:

- Date
- Math
- String, Number, Boolean
- RegExp
- window (Global Object)

JavaScript String Object

Of the above objects, the most widely used one is the String object. Objects are nothing but special kind of data. Each object has Properties and Methods associated with it. *property* is the value that is tagged to the object. For example let us consider one of the properties associated with the most popularly used String object - the *length* property. *Length* property of the string object returns the length of the string that is in other words the number of characters present in the string.

General syntax of using the *length* property of the string object is as below:

`variablename.length`

Here *variablename* is the name of the variable to which the string is assigned and *length* is the keyword.

For example consider the JavaScript below:

```
<html>
  <body>
    <script type="text/javascript">
      var exf="Welcome"
      document.write(exf.length)

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

The output of the above is

7

Method of an Object:

Method of an object refers to the actions that can be performed on the object. For example in String Object there are several methods available in JavaScript.

Example to understand how method can be used in an Object.

In the example below, we have used *toUpperCase* method of String object.

```
<html>
  <body>
    <script type="text/javascript">
      var exf="Welcome"
      document.write(exf.toUpperCase())

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

The output of the above script is

WELCOME

In the above script since the method *toUpperCase* is applied to the string object *exf* which has value initialized as *Welcome* all letters get converted as upper case and hence the output is as above.

Purpose of String Object in JavaScript:

The main purpose of String Object in JavaScript is for storing text. General method of using String Object is to declare a variable and assign a string, in other words a text to the variable.

```
var exf="Welcome"
```

assigns the text *Welcome* to the variable *exf* defined.

String Object Methods

Method	Description
charAt()	Returns the character at the specified index
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
match()	Searches for a match between a regular expression and a string, and returns the matches
replace()	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring
search()	Searches for a match between a regular expression and a string, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toUpperCase()	Converts a string to uppercase letters

valueOf()	Returns the primitive value of a String object
-----------	--

JavaScript Date Object

Usage of Date Object:

Date object of Java Script is used to work with date and times. General syntax for defining Date object in Java Script is as follows:

```
var variablename=new Date( )
```

In the above *new* is a keyword which creates an instance of object and *Date()* defines *variablename* as Date Object.

For example:

```
var exf=new Date( )
```

In the above example, variable *exf* is defined as Date object which has current date and time as its initial value.

Methods of Date Object:

Some of the methods available with Date object are:

setSeconds() - Sets the seconds for a specified date according to local time.
 setMinutes() - Sets the minutes for a specified date according to local time.
 setHours() - Sets the hours for a specified date according to local time. setDate() -
 Sets the day of the month for a specified date according to local time. setMonth() -
 Sets the month for a specified date according to local time.
 setYear() - Sets the year (deprecated) for a specified date according to local time.
 setFullYear() - Sets the full year for a specified date according to local time.
 toString() - Returns a string representing the specified Date object.
 getSeconds() - Returns seconds in the specified date according to local time.
 getMinutes() - Returns minutes in the specified date according to local time.
 getHours() - Returns hour in the specified date according to local time. getDay()
 - Returns day of the week for a specified date according to local time
 getDate() - Returns day of the month for a specified date according to local time.
 getMonth() - Returns month in the specified date according to local time. getYear()
 - Returns year (deprecated) in the specified date according to local time.
 getFullYear() - Returns year of the specified date according to local time.

Example for usage of Date Object methods mentioned above:

```
var exf=new Date()
```

```
exf.setFullYear(2020,0,20)
```

As we have seen *setFullYear()* is used for Setting the full year for a specified date according to local time. In the above example the Date object *exf* is set to the specific date and year *20th January 2020*

Example for using methods of Date Object

```
<html>
<body>
  <script type="text/javascript">
    var exforsys=new Date();
    var currentDay=exforsys.getDate();
    var currentMonth=exforsys.getMonth() + 1;
    var currentYear=exforsys.getFullYear();
    document.write(currentMonth + "/" + currentDay +
      "/" + currentYear);

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

Output of the above program is:

11/15/2006

JavaScript Math Object

Usage of Math Object:

JavaScript *Math* object is used to perform mathematical tasks. But unlike the *String* and the *Date* object which requires defining the object, *Math* object need not be defined. *Math* object in JavaScript has two main attributes:

- Properties
- Methods

Properties of Math Object:

The JavaScript has eight mathematical values and this can be accessed by using the *Math* Object. The eight mathematical values are:

- E
- PI
- square root of 2 denoted as SQRT2
- square root of 1/2 denoted as SQRT1_2
- natural log of 2 denoted as LN2
- natural log of 10 denoted as LN10
- base-2 log of E denoted as LOG2E
- base-10 log of E denoted as LOG10E

The way of accessing these values in JavaScript is by using the word *Math* before these values namely as

- Math.E
- Math.LOG10E and so on

Methods of Math Object:

There are numerous methods available in JavaScript for *Math* Object. Some of them are mentioned below namely:

- `abs(x)` - Returns absolute value of x.
- `acos(x)` - Returns arc cosine of x in radians.
- `asin(x)` - Returns arc sine of x in radians.
- `atan(x)` - Returns arc tan of x in radians.
- `atan2(y, x)` - Counterclockwise angle between x axis and point (x,y).
- `ceil(x)` - Returns the smallest integer greater than or equal to x. (round up).
- `cos(x)` - Returns cosine of x, where x is in radians.
- `exp(x)` - Returns e^x
- `floor(x)` - Returns the largest integer less than or equal to x. (round down)
- `log(x)` - Returns the natural logarithm (base E) of x.
- `max(a, b)` - Returns the larger of a and b.
- `min(a, b)` - Returns the lesser of a and b.
- `pow(x, y)` - Returns x^y
- `random()` - Returns a pseudorandom number between 0 and 1.
- `round(x)` - Rounds x up or down to the nearest integer. It rounds .5 up.
- `sin(x)` - Returns the Sin of x, where x is in radians.
- `sqrt(x)` - Returns the square root of x.
- `tan(x)` - Returns the Tan of x, where x is in radians.

Example for *Math* Object methods mentioned above:

```
<html>
<body>
  <script type="text/javascript">
    document.write(Math.round(5.8))

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

The output of the above program is

6

This is because the *round()* method rounds the number given in argument namely here 5.8 to the nearest integer. It rounds .5 up which gives 6.

Another example for using *Math* Object in JavaScript.

```
<html>
<body>
  <script type="text/javascript">
    document.write(Math.max(8,9) + "<br />")
    document.write(Math.max(-5,3) + "<br />")
    document.write(Math.max(-2,-7) + "<br />")

  </script>
</body>
```

```
</html>
```

Output of the above program is

```
9
3
-2
```

The above example uses the *max()* method of the *Math* object which returns the largest of the two numbers given in arguments of the max method.

JavaScript Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

Boolean Object Methods

Method	Description
toString()	Converts a Boolean value to a string, and returns the result
valueOf()	Returns the primitive value of a Boolean object

Number Object

The Number object is an object wrapper for primitive numeric values.

Number objects are created with new Number().

Syntax

```
var num = new Number(value);
```

Number Object Methods

Method	Description
toExponential(x)	Converts a number into an exponential notation
toFixed(x)	Formats a number with x numbers of digits after the decimal point
toPrecision(x)	Formats a number to x length
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object

String Object

The String object is used to manipulate a stored piece of text.

String objects are created with new String().

Syntax


```
var txt = new String(string);
```

or more simply:

```
var txt = string;
```

Window Object

The window object represents an open window in a browser.

If a document contains frames (<frame> or <iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

Window Object Methods

Method	Description
alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
createPopup()	Creates a pop-up window
focus()	Sets focus to the current window
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height
scroll()	
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

JavaScript RegExp Object

Regular expressions are used to do sophisticated pattern matching, which can often be helpful in form validation. For example, a regular expression can be used to check whether an email address entered into a form field is syntactically correct. JavaScript supports Perl-compatible regular expressions.

There are two ways to create a regular expression in JavaScript:

1. Using literal syntax

```
var reExample = /pattern/;
```

2. Using the RegExp() constructor

```
var reExample = new RegExp("pattern");
```

Assuming you know the regular expression pattern you are going to use, there is no real difference between the two; however, if you don't know the pattern ahead of time (e.g, you're retrieving it from a form), it can be easier to use the RegExp() constructor.

JavaScript's Regular Expression Methods

The regular expression method in JavaScript has two main methods for testing strings: test() and exec().
The exec() Method

The exec() method takes one argument, a string, and checks whether that string contains one or more matches of the pattern specified by the regular expression. If one or more matches is found, the method returns a result array with the starting points of the matches. If no match is found, the method returns null.

The test() Method

The test() method also takes one argument, a string, and checks whether that string contains a match of the pattern specified by the regular expression. It returns true if it does contain a match and false if it does not. This method is very useful in form validation scripts. The code sample below shows how it can be used for checking a social security number. Don't worry about the syntax of the regular expression itself. We'll cover that shortly.

Code Sample: RegularExpressions for validating social security number

```
<html>
<head>
<script type="text/javascript">
var exp = /^[0-9]{3}[\- ]?[0-9]{2}[\- ]?[0-9]{4}$/;

function f1(ssn)
{
    if (exp.test(ssn)) { alert("VALID SSN"); }
    else { alert("INVALID SSN"); }
}
</script>
</head>
<body>
<form name="f1">
    <input type="text" name="t1" />
    <input type="button" value="Check" onclick="f1(this.f1.t1.value);" />
</form>
</body>
</html>
```

Code Explanation

Let's examine the code more closely:

1. First, a variable containing a regular expression object for a social security number is declared.

```
var exp = /^[0-9]{3}[\- ]?[0-9]{2}[\- ]?[0-9]{4}$/;
```

2. Next, a function called f1() is created. This function takes one argument: ssn, which is a string. The function then tests to see if the string matches the regular expression pattern by passing it to the regular expression object's test() method. If it does match, the function alerts "VALID SSN". Otherwise, it alerts "INVALID SSN".

```
function f1(ssn)
{
  if (exp.test(ssn)) { alert("VALID SSN"); } else
  { alert("INVALID SSN"); }
}
```

3. A form in the body of the page provides a text field for inserting a social security number and a button that passes the user-entered social security number to the f1() function.

```
<form >
  <input type="text" name="t1" />
  <input type="button" value="Check" onclick="checkSsn(this.form.ssn.value);" />
</form>
```

Flags

Flags appearing after the end slash modify how a regular expression works.

- * The i flag makes a regular expression case insensitive. For example, /aeiou/i matches all lowercase and uppercase vowels.

- * The g flag specifies a global match, meaning that all matches of the specified pattern should be returned.

Regular Expression Syntax

A regular expression is a pattern that specifies a list of characters.

Start and End : ^ \$

A caret (^) at the beginning of a regular expression indicates that the string being searched must start with this pattern.

- * The pattern ^foo can be found in "food", but not in "barfood".

A dollar sign (\$) at the end of a regular expression indicates that the string being searched must end with this pattern.

- * The pattern foo\$ can be found in "curfoo", but not in "food".

Number of Occurrences : ? + * {}

The following symbols affect the number of occurrences of the preceding character: ?, +, *, and {}.

A questionmark (?) indicates that the preceding character should appear zero or one times in the pattern.

- * The pattern foo? can be found in "food" and "fod", but not "faod".

A plus sign (+) indicates that the preceding character should appear one or more times in the pattern. *

The pattern fo+ can be found in "fod", "food" and "foood", but not "fd".

A asterisk (*) indicates that the preceding character should appear zero or more times in the pattern.

- * The pattern `fo*d` can be found in "fd", "fod" and "food".

Curly brackets with one parameter (`{n}`) indicate that the preceding character should appear exactly `n` times in the pattern.

- * The pattern `fo{3}d` can be found in "foood", but not "food" or "foood".

Curly brackets with two parameters (`{n1,n2}`) indicate that the preceding character should appear between `n1` and `n2` times in the pattern.

- * The pattern `fo{2,4}d` can be found in "food", "foood" and "foood", but not "fod" or "fooooo".

Curly brackets with one parameter and an empty second parameter (`{n,}`) indicate that the preceding character should appear at least `n` times in the pattern.

- * The pattern `fo{2,}d` can be found in "food" and "fooooo", but not "fod".

Common Characters: . \d \D \w \W \s \S

A period (`.`) represents any character except a newline.

- * The pattern `fo.d` can be found in "food", "foad", "fo9d", and "fo*d".

Backslash-d (`\d`) represents any digit. It is the equivalent of `[0-9]`.

- * The pattern `fo\dd` can be found in "fo1d", "fo4d" and "fo0d", but not in "food" or "fodd".

Backslash-D (`\D`) represents any character except a digit. It is the equivalent of `[^0-9]`.

- * The pattern `fo\Dd` can be found in "food" and "foad", but not in "fo4d".

Backslash-w (`\w`) represents any word character (letters, digits, and the underscore (`_`)).

- * The pattern `fo\wd` can be found in "food", "fo_d" and "fo4d", but not in "fo*d".

Backslash-W (`\W`) represents any character except a word character.

- * The pattern `fo\Wd` can be found in "fo*d", "fo@d" and "fo.d", but not in "food".

Backslash-s (`\s`) represents any whitespace character (e.g, space, tab, newline, etc.).

- * The pattern `fo\s d` can be found in "fo d", but not in "food". Backslash-

S (`\S`) represents any character except a whitespace character.

- * The pattern `fo\Sd` can be found in "fo*d", "food" and "fo4d", but not in "fo d".

Grouping: []

Square brackets (`[]`) are used to group options.

- * The pattern `f[aeiou]d` can be found in "fad" and "fed", but not in "food", "faed" or "fd".
- * The pattern `f[aeiou]{2}d` can be found in "faed" and "feod", but not in "fod", "fed" or "fd".

Negation : ^

When used after the first character of the regular expression, the caret (^) is used for negation.

- * The pattern `f[^aeiou]d` can be found in "fqd" and "f4d", but not in "fad" or "fed".

Subpatterns: ()

Parentheses () are used to capture subpatterns.

- * The pattern `f(oo)?d` can be found in "food" and "fd", but not in "fod".

Alternatives: |

The pipe (|) is used to create optional patterns.

- * The pattern `foo$/^bar` can be found in "foo" and "bar", but not "foobar".

Escape Character : \

The backslash (\) is used to escape special characters.

- * The pattern `fo\d` can be found in "fo.d", but not in "food" or "fo4d".

A more practical example has to do matching the delimiter in social security numbers. Examine the following regular expression.

`^\d{3}([ - ]?)\d{2}([ - ]?)\d{4}$`

Within the caret (^) and dollar sign (\$), which are used to specify the beginning and end of the pattern, there are three sequences of digits, optionally separated by a hyphen or a space. This pattern will be matched in all of following strings (and more).

- * 123-45-6789
- * 123 45 6789
- * 123456789
- * 123-45 6789
- * 123 45-6789
- * 123-456789

The last three strings are not ideal, but they do match the pattern. Back references can be used to make sure that the second delimiter matches the first delimiter. The regular expression would look like this.

`^\d{3}([ - ]?)\d{2}\1\d{4}$`

The `\1` refers back to the first subpattern. Only the first three strings listed above match this regular expression.

Form Validation with Regular Expressions

Regular expressions make it easy to create powerful form validation functions. Take a look at the following example.

Code Sample: Login.html

```
<html>
<head>
<script type="text/javascript">
```

```

var RE_EMAIL = /^(w+[\-\.\])*\w+@(\w+\.)+[A-Za-z]+$;/
var RE_PASSWORD = /^[A-Za-z\d]{6,8}$/;

function validate()
{
  var email = form.Email.value;
  var password = form.Password.value;
  var errors = [];
  if (!RE_EMAIL.test(email)) { alert( "You must enter a valid email address."); }
  if (!RE_PASSWORD.test(password)) { alert( "You must enter a valid password."); }
}

</script>
</head>
<body>
  <form name="form">
    Email: <input type="text" name="Email" />
    Password: <input type="password" name="Password" />
    *Password must be between 6 and 10 characters and can only contain letters and digits.

    <input type="submit" value="Submit" onclick="Validate();" />
    <input type="reset" value="Reset Form" />
  </p>
</form>
</body>
</html>

```

Code Explanation

This code starts by defining regular expressions for an email address and a password. Let's break each one down.

```
var RE_EMAIL = /^(w+[\-\.\])*\w+@(\w+\.)+[A-Za-z]+$;/
```

1. The caret (^) says to start at the beginning. This prevents the user from entering invalid characters at the beginning of the email address.
2. (w+[\-\.\])* allows for a sequence of word characters followed by a dot or a dash. The * indicates that the pattern can be repeated zero or more times. Successful patterns include "ndunn.", "ndunn-", "nat.s.", and "nat-s-".
3. \w+ allows for one or more word characters.
4. @ allows for a single @ symbol.
5. (\w+\.)+ allows for a sequence of word characters followed by a dot. The + indicates that the pattern can be repeated one or more times. This is the domain name without the last portion (e.g, without the "com" or "gov").
6. [A-Za-z]+ allows for one or more letters. This is the "com" or "gov" portion of the email address.
7. The dollar sign (\$) says to end here. This prevents the user from entering invalid characters at the end of the email address.

```
var RE_PASSWORD = /^[A-Za-z\d]{6,8}$/;
```

1. The caret (^) says to start at the beginning. This prevents the user from entering invalid characters at the beginning of the password.
2. [A-Za-z\d]{6,8} allows for a six- to eight-character sequence of letters and digits.
3. The dollar sign (\$) says to end here. This prevents the user from entering invalid characters at the end of the password.

Exercises:

1. Construct a reg exp to validate a text field which should be used to accept only a string composed by 3 letters, one space, 6 numbers, a "-" and a number such as MJHJ 123456-6

Ans: `/^[A-Za-z]{4}\s\d{6}\-\d{1}$/`

2. Write regular expressions to check for:
 1. Proper Name
 - o starts with capital letter
 - o followed by one or more letters or apostrophes
 - o may be multiple words (e.g, "New York City")
 2. Initial
 - o zero or one capital letters
 3. State
 - o two capital letters
 4. Postal Code
 - o five digits (e.g, "02138")
 - o possibly followed by a dash and four digits (e.g, "-1234")
 5. Username
 - o between 6 and 15 letters or digits
3. Add validation to check the following fields:
 1. first name
 2. middle initial
 3. last name
 4. city
 5. state
 6. zip
 7. username
3. Test your solution in a browser.

Document Object

Each HTML document loaded into a browser window becomes a Document object. The Document object provides access to all HTML elements in a page, from within a script.

Document Object Methods

Method	Description
<code>close()</code>	Closes the output stream previously opened with <code>document.open()</code>
<code>getElementById()</code>	Accesses the first element with the specified id
<code>getElementsByName()</code>	Accesses all elements with a specified name
<code>getElementsByTagName()</code>	Accesses all elements with a specified tagname
<code>open()</code>	Opens an output stream to collect the output from <code>document.write()</code> or <code>document.writeln()</code>
<code>write()</code>	Writes HTML expressions or JavaScript code to a document
<code>writeln()</code>	Same as <code>write()</code> , but adds a newline character after each statement

Arrays

It describes the JavaScript array object including parameters, properties, and methods.

Parameters

- * `arrayLength`
- * `elementN` - Array element list of values

Properties

- * index
- * input
- * length - The quantity of elements in the object.
- * prototype - For creating more properties.

Methods

* chop() - Used to truncate the last character of all strings that are part of an array. This method is not defined so it must be written and included in your code.

```
var exclamations = new Array("Look out!", "Duck!")
exclamations.chop()
```

Causes the values of exclamations to become:

Look out
Duck

- * concat()
- * grep(searchstring) - Takes an array and returns those array element strings that contain matching strings. This method is not defined so it must be written and included in your code.

```
words = new Array("limit", "lines", "finish", "complete", "In", "Out")
inwords = words.grep("in")
```

The array, inwords, will be:

lines, finish

- * join(delimiter) - Puts all elements in the array into a string, separating each element with the specified delimiter.

```
words = new Array("limit", "lines", "finish", "complete", "In", "Out")
var jwords = words.join(";")
```

The value of the string jwords is:

limit;lines;finish;complete;In;Out

- * pop() - Pops the last string off the array and returns it. This method is not defined so it must be written and included in your code.

```
words = new Array("limit", "lines", "finish", "complete", "In", "Out")
var lastword = words.pop()
```

The value of the string lastword is:

Out

- * push(strings) - Strings are placed at the end of the array. This method is not defined so it must be written and included in your code.

```
words = new Array("limit", "lines", "finish")
words.push("complete", "In", "Out")
```


The array, words, will be:

limit, lines, finish, complete, In, Out

* reverse() - Puts array elements in reverse order.

```
words = new Array("limit","lines","finish","complete","In","Out")
words.reverse()
```

The array, words, will be:

Out, In, complete, finish, lines, limit

* shift() - Decreases array element size by one by shifting the first element off the array and returning it. This method is not defined so it must be written and included in your code.

```
words = new Array("limit","lines","finish","complete","In","Out")
word = words.shift()
```

The array, words, will be:

In, complete, finish, lines, limit

The string word will be:

Out

*sort() - Sorts the array elements in dictionary order or using a compare function passed to the method.

```
words = new Array("limit","lines","finish","complete","In","Out")
word = words.sort()
```

The value of words becomes:

In,Out,complete,finish,limit,lines

* splice() - It is used to take elements out of an array and replace them with those specified. In the below example the element starting at element 3 is removed, two of them are removed and replaced with the specified strings. The value returned are those values that are replaced. This method is not defined so it must be written and included in your code.

```
words = new Array("limit","lines","finish","complete","In","Out")
words1 = words.splice(3, 2, "done", "On")
```

The value of words becomes:

limit, lines, finish, done, On, Out

The value of words1 is set to:

complete, In

*split(delimiter) - Splits a string using the delimiter and returns an array.

```
words = new String("limit;lines;finish;complete;In;Out")
```

```
var swords = words.split(";")
```

The values in the array swords is:

limit, lines, finish, complete, In, Out

* unshift() - Places elements at the start of an array

```
words = new Array("finish","complete","In","Out")
```

```
word = words.unshift("limit","lines")
```

The array, words, will be:

limit, lines, finish, complete, In, Out

Form validation

Form validation is the process of checking that a form has been filled in correctly before it is processed. For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form. Form validation is usually done with JavaScript embedded in the Web page

[Validate text field to accept e-mail id](#)

```

<html>
<head>
<script>
function f1()
{
    var email=myForm.t1.value;
    if(email=="") alert("Enter E-mail ID"); // if no input
    if(email.indexOf("@")==-1) alert("Invalid Id"); // if input is: raj.com
    if(!(email.indexOf("@")==email.lastIndexOf("@"))) alert("Invalid Id"); // if input is:raj@kumar@com
    if(email.indexOf(".",0)==-1) alert("Invalid Id"); // if input is:raj@yahoo.com
    if(!(email.indexOf("@")<email.lastIndexOf("."))) alert("Invalid Id"); // if input is: raj.kumar@yahoo.com
}
</script>
</head>
<body>
<form name="myForm">
Email ID:<input type="text" name="t1" />
<input type="button" value="click" onclick=f1 () />
</form>
</body>
</html>

```

Output of the program

Age:

The above program will accept the input in any one of the following valid form
 raj@yahoo.com raj.kumar@yahoo.com raj.k@yahoo.co.in

validate text field to accept name

```

<html>
<head>
<script>
function f1()
{
    var name=myForm.t1.value;
    if(name=="") alert("Enter name"); // if the input is empty
    for(var i=0; i<name.length; i++)
    {
        if ( (! (name.charAt(i)>='a' && name.charAt(i) <='z' ) ||
            (name.charAt(i)>='A' && name.charAt(i) <='Z' ) )) // if input is: raj321 or 321raj
        {
            alert("not a valid name");
            break;
        }
    }
}
</script>
</head>
<body>
<form name="myForm">
Enter Name:<input type="text" name="t1" />
<input type="button" value="click" onclick=f1 () />
</form>
</body>
</html>

```

Output of the program

Enter Name:

The above program will accept the input only in the following valid form
Rajkumar

Validate text field to accept an age

```
<html>
<head>
<script>
function fl()
{
    var age=myForm.tl.value;
    if(age=="") alert("Empty field"); // if the input is empty
    if(age.length>3) alert("invalid"); // if the input is:1015
    if(isNaN(age)) alert("invalid"); //if the input is: abc
}
</script>
</head>
<body>
<form name="myForm">
Enter Age:<input type="text" name="t1" />
<input type="button" value="click" onclick=fl() />
</form>
</body>
</html>
```

Output of the program

Enter Age:

The above program will accept the input only in any one of the following valid form
25 5 101

Validate a checkbox

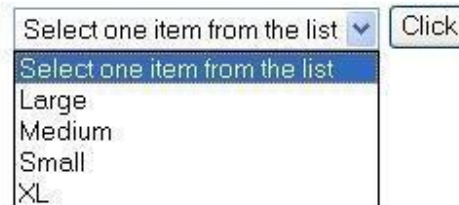
```
<html>
<head>
<script>
function fl()
{
    if (!myForm.c1[0].checked && !myForm.c1[1].checked &&
        !myForm.c1[2].checked )
    {
        alert("Select any one size");
    }
}
</script>
</head>
<body>
<form name="myForm">
<input type="radio" name="c1" /> Large
<input type="radio" name="c1" /> XL
<input type="radio" name="c1" /> XXL
<input type="button" value="click" onclick=fl() />
</form>
</body>
</html>
```

☐ Large ☐ XL ☐ XXL

Validate form selection

```
<html>
<head>
<script>
function fl()
{
    var sel=myForm.s1.selectedIndex;
    if(sel==0) alert("S elect one item");

}
</script>
</head>
<body>
<form name="myForm">
<select name="s1">
<OPTION SELECTED>Select one item from the list
<OPTION VALUE="one">Large
<OPTION VALUE="two">Medium
<OPTION VALUE="three">Small
<OPTION VALUE="four">XL
</option>
</select>
<input type="button" value="Click" onclick=fl() />
</form>
</body>
</html>
```



The screenshot shows a web form with a dropdown menu and a button. The dropdown menu is open, displaying the following options: "Select one item from the list" (highlighted in blue), "Large", "Medium", "Small", and "XL". The button is labeled "Click".

UNIT - III

Host Objects

JavaScript supports three types of objects: native, host, and user-defined. Native objects are objects supplied by the JavaScript language. String, Boolean, Math, and Number are examples of native objects.

Host objects are JavaScript objects that provide special access to the host environment. They are provided by the browser for the purpose of interaction with the loaded document. In a browser environment,

1. window
2. document

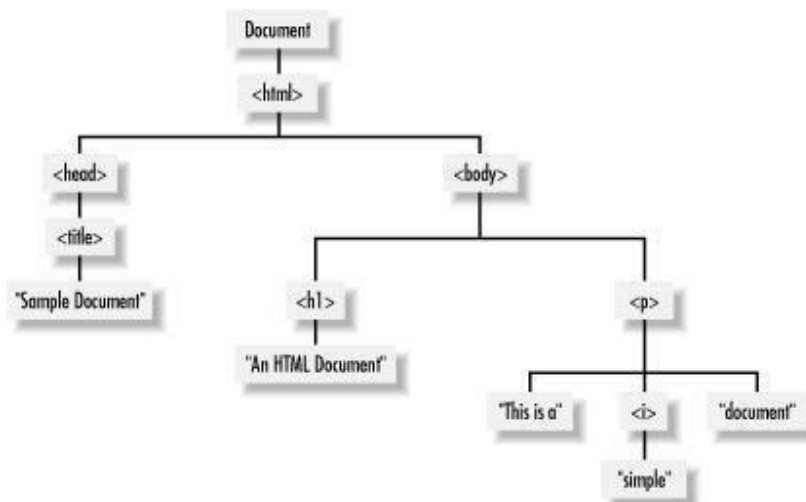
objects are host objects. Several other browser host objects are informal, *de facto* standards. They are: alert, prompt, confirm.

DOM

- The Document Object Model (DOM) is an API that allows programs to interact with HTML (or XML) documents
- The primary function of the Document Object Model is to view, access, and change the structure of an HTML document separate from the content contained within it.
- The DOM will provide you with methods and properties to retrieve, modify, update, and delete parts of the document you are working on. The properties of the Document Object Model are used to describe the web page or document and the methods of the Document Object Model are used for working with parts of the web page.
- In DOM, HTML document is represented in tree like structure. It constructs a hierarchical tree structure for a HTML document to traverse and to manipulate the document.
- For example,

```
<html>
<head>
  <title>Sample Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <i>simple</i> document.
</body>
</html>
```

The DOM representation of this document is as follows:



The node directly above a node is the *parent* of that node. The nodes one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node.

Types of nodes

- There are many types of nodes in the DOM document tree that specifies what kind of node it is. Every Object in the DOM document tree has properties and methods defined by the Node host object.

The following table lists the non method properties of Node object.

TABLE 5.2: Non-method properties of `Node` instances.

Property	Description
<code>nodeType</code>	Number representing the type of node (<code>Element</code> , <code>Comment</code> , etc.).
<code>nodeName</code>	String providing a name for this <code>Node</code> (form of name depends on the <code>nodeType</code> ; see text).
<code>parentNode</code>	Reference to object that is this node's parent.
<code>childNodes</code>	Acts like a read-only array containing this node's child nodes. Has <code>length</code> 0 if this node has no children.
<code>previousSibling</code>	Previous sibling of this node, or <code>null</code> if no previous sibling exists.
<code>nextSibling</code>	Next sibling of this node, or <code>null</code> if no next sibling exists.
<code>attributes</code>	Acts like a read-only array containing <code>Attr</code> instances representing this node's attributes.

The following table lists the node types commonly encountered in HTML documents and the `nodeType` value for each one.

Node Type	<code>nodeType</code> constant	<code>nodeType</code> value
Element	<code>Node.ELEMENT_NODE</code>	1
Text	<code>Node.TEXT_NODE</code>	3
Document	<code>Node.DOCUMENT_NODE</code>	9
Comment	<code>Node.COMMENT_NODE</code>	8
DocumentFragment	<code>Node.DOCUMENT_FRAGMENT_NODE</code>	11
Attr	<code>Node.ATTRIBUTE_NODE</code>	2

The following table lists the method properties of `Node` object.

TABLE 5.4: Method properties of `Node` instances.

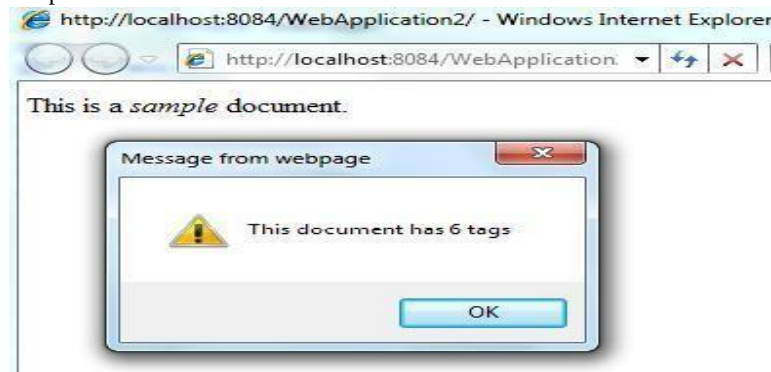
Method	Functionality
<code>hasAttributes()</code>	Returns Boolean indicating whether or not this node has attributes.
<code>hasChildNodes()</code>	Returns Boolean indicating whether or not this node has children.
<code>appendChild(Node)</code>	Adds the argument <code>Node</code> to the end of the list of children of this node.
<code>insertBefore(Node, Node)</code>	Adds the first argument <code>Node</code> in the list of children of this node immediately before the second argument <code>Node</code> (or at end of child list if second argument is <code>null</code>).
<code>removeChild(Node)</code>	Removes the argument <code>Node</code> from this node's list of children.
<code>replaceChild(Node, Node)</code>	In the list of children of this node, replace the second argument <code>Node</code> with the first.

Traversing a Document: Counting the number of Tags

The DOM represents an HTML document as a tree of Node objects. With any tree structure, one of the most common things to do is traverse the tree, examining each node of the tree in turn. The following program shows one way to do this.

```
<html>
<head>
<script>
function countTags(n)
{
    // n is a Node
    var numtags = 0;        // Initialize the tag counter
    if (n.nodeType == 1 ) // Check if n is an Element
        numtags++;        // Increment the counter if so
    var children = n.childNodes; // Now get all children of n
    for(var i=0; i < children.length; i++)
    { // Loop through the children
        numtags += countTags(children[i]); // Recurse on each one
    }
    return numtags;        // Return the total number of tags
}
</script>
</head>
<body onload="alert('This document has ' + countTags(document) + ' tags')">
This is a <i>sample</i> document.
</body>
</html>
```

Output



Finding Specific Elements in a Document

The ability to traverse all nodes in a document tree gives us the power to find specific nodes. When programming with the DOM API, it is quite common to need a particular node within the document or a list of nodes of a specific type within the document.

You can use `getElementById()` and `getElementsByTagName()` methods of Document Object to obtain a list of any type of HTML element. For example, to find all the tables within a document, you'd do this:

```
var tables = document.getElementsByTagName("table");
```

This code finds <table> tags and returns elements in the order in which they appear in the document.

getElementById() to find a specific element whereas getElementsByName() returns an array of elements rather than a single element.

The following program illustrates this.

```
<html>
  <head>
    <script type="text/javascript">
      function f1()
      {
        alert(document.getElementById("p1").nodeName);
      }
    </script>
  </head>
  <body>
    <p id="p1"> Program is coded to find paragraph element is present in the document or not using its
      id attribute</p>
    <input type="button" value="Find" onclick="f1();" />

  </body>
</html>
```

Output



In the above program the method getElementById() finds the specific element and nodeName is used property return the specific element name.

Modifying a Document: Reversing the nodes of a document

DOM API lies in the features that allow you to use JavaScript to dynamically modify documents. The following examples demonstrate the basic techniques of modifying documents and illustrate some of the possibilities.

The following example includes a JavaScript function named reverse(), a sample document, and an HTML button that, when pressed, calls the reverse() function, passing it the node that represents the <body> element of the document. The reverse() function loops backward through the children of the supplied node and uses the removeChild() and appendChild() methods of the Node object to reverse the order of those children.

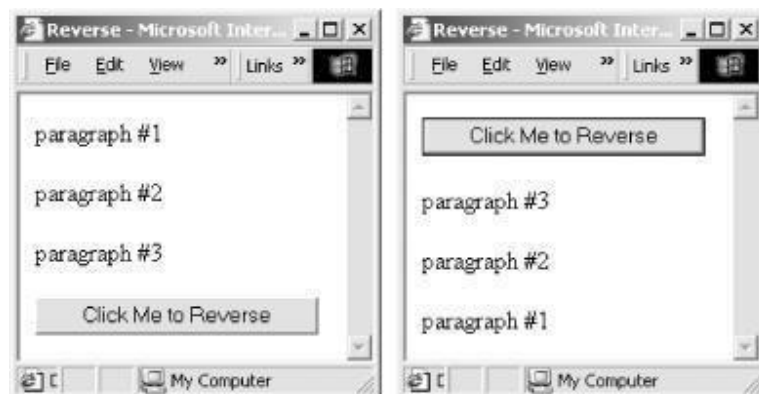
```
<html>
  <head><title>Reverse</title>
  <script>
    function reverse(n) {      // Reverse the order of the children of Node n
```

```

    var kids = n.childNodes; // Get the list of children
    var numkids = kids.length; // Figure out how many children there are
    for(var i = numkids-1; i >= 0; i--) { // Loop backward through the children
        var c = n.removeChild(kids[i]); // Remove a child
        n.appendChild(c); // Put it back at its new position
    }
}
</script>
</head>
<body>
    <pre>
        paragraph #1
        paragraph #2
        paragraph #3
    </pre>
    <form>
        <input type="button" value="Click Me to reverse" onclick="reverse(document.body);"/>
    </form>
</body>
</html>

```

when the user clicks the button, the order of the paragraphs and of the button are reversed.



Changing element Style

The following program illustrates how to change the element style using DOM properties and methods.

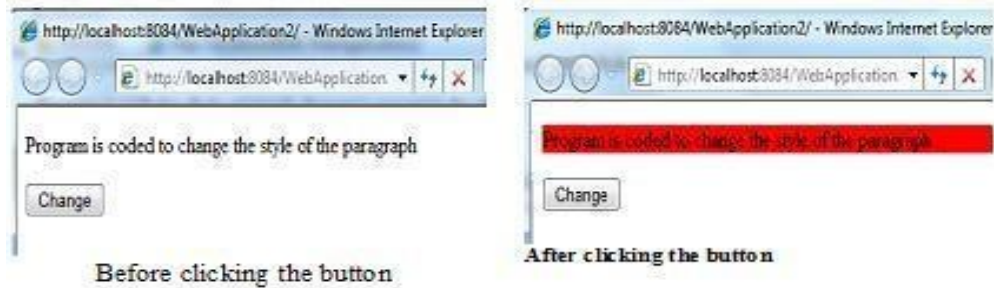
```

<html>
  <head>
    <script type="text/javascript">
      function f1()
      {
        document.getElementById("p1").style.backgroundColor="red";
      }
    </script>
  </head>
  <body>
    <p id="p1"> Program is coded to change the style of the paragraph</p>
    <input type="button" value="Change" onclick="f1();" />
  </body>
</html>

```

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

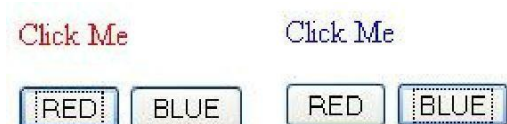
The method `getElementById()` gets the paragraph element in the document and the property style is used to change the background color of the paragraph to “red” as shown below.



Changing element style

```
<html>
<head>
<script type="text/javascript">
function f1()
{
    var o=document.getElementById("p1");
    o.style.color="red";
}
function f2()
{
    var o=document.getElementById("p1");
    o.style.color="blue";
}
</script>
</head>
<body>
<p id="p1"> Click
    Me
</p>
<form>
<input type=button id="b1" value="RED" onclick=f1() />
<input type=button id="b2" value="BLUE" onclick=f2() />
</form>
</body>
</html>
```

Output



Changing HTML Content

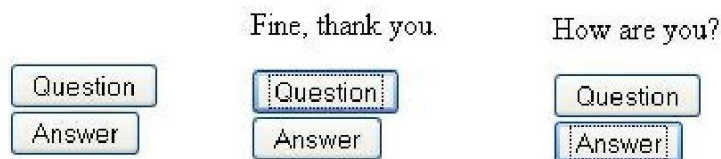
This page shows an example of how to change a HTML page's content

```

<html>
<head>
<script type="text/javascript">
function f1()
{
    document.getElementById("p1").childNodes[0].nodeValue="Fine, thank you.";
}
function f2()
{
    document.getElementById("p1").childNodes[0].nodeValue="How are you?";
}
</script>
</head>
<body id="p1">
<pre>
<input type="button" id="b1" value="Question" onclick=f1() />
<input type="button" id="b2" value="Answer" onclick=f2() />
</pre>
</body>
</html>

```

After pressing the Question button, it adds the content, How are you?" to the HTML document and after pressing the Answer button, it replaces the content "How are you?" with "Fine, thank you"



Removing Element from HTML documents

```

<html>
<head>
<script type="text/javascript">
function f1()
{
    var node=document.getElementById("p1");
    node.removeChild(node.childNodes[0]);
}
</script>
</head>
<body >
<pre id="p1"><input type="button" id="b1" value="Question" />
<input type="button" id="b2" value="Remove" onclick=f1() />
Example for Removing an element from HTML document. </pre>

</body>
</html>

```

After pressing the "Remove" button, the element "Question" is removed from the document.



Example for Removing an element from HTML document.



Example for Removing an element from HTML document.

Server-side Programming: Servlet

The combination of

- HTML
- JavaScript
- DOM

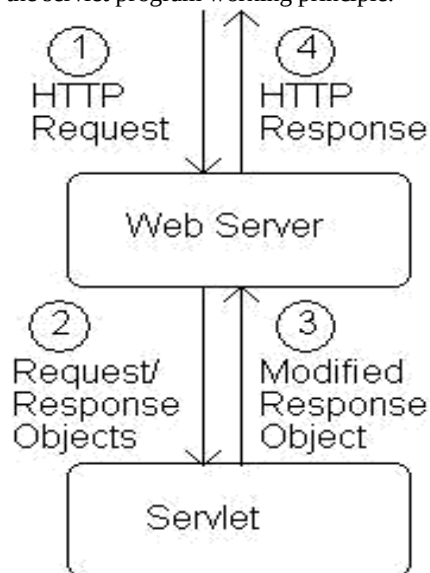
is sometimes referred to as Dynamic HTML (DHTML). Web pages that include scripting are often called dynamic pages. Similarly, web server response can be static or dynamic

- Static: **HTML document is retrieved** from the file system by the server and the same returned to the client.
- Dynamic: In server, a **HTML document is generated** by a program in response to an HTTP request

Java servlets are one technology for producing dynamic server responses. Servlet is a class instantiated by the server to produce a dynamic response.

Servlet Overview

The following figure illustrates the servlet program working principle.



1. When server starts it instantiates servlets
2. Server receives HTTP request, determines need for dynamic response
3. Server selects the appropriate servlet to generate the response, creates request/response objects, and passes them to a method on the servlet instance
4. Servlet adds information to response object via method calls
5. Server generates HTTP response based on information stored in response object

Types of Servlet

- Generic Servlet
- HttpServlet

Servlets vs. Java Applications

- ♦ Servlets do not have a main() method
- ♦ Entry point to servlet code is via call to a method doGet() /doPost()
- ♦ Servlet interaction with end user is indirect via request/response object APIs
- ♦ Primary servlet output is typically HTML

Running Servlets

1. Compile servlet (make sure that JWSDP libraries are on path)
2. Copy .class file to **shared/classes** directory
3. (Re)start the Tomcat web server
4. If the class is named ServletHello, browse to
http://localhost:8080/servlet/ServletHello

What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

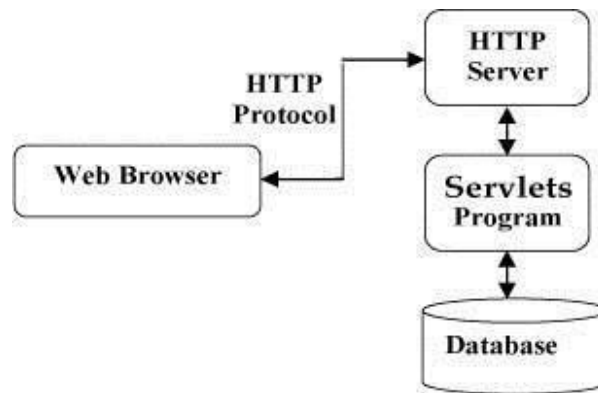
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture:

Following diagram shows the position of Servlets in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks:

1. Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
2. Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
3. Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
4. Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
5. Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.
 - The servlet calls **service()** method to process a client's request.
-

- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {
    // Initialization code...
}
```

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
public void service(ServletRequest request,
    ServletResponse response)
    throws ServletException, IOException{
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here are the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

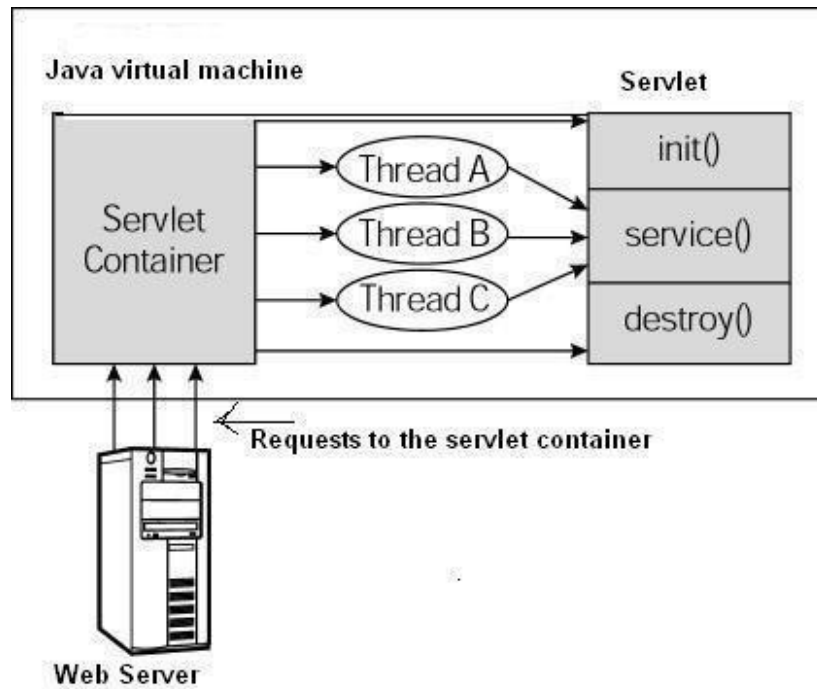
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
    // Finalization code...
}
```

Architecture Diagram:

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Structure of a servlet program

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class NewServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");           // content type of the response
        PrintWriter out = response.getWriter();         // used to create a response as a Html doc
        try {
            out.println("<html>");
            -----
            out.println("</html>");
        } catch (Exception e) {}
    }
}
```

Servlets - Examples

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend `javax.servlet.http.HttpServlet`, an abstract

class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code for Hello World:

Following is the sample source code structure of a servlet example to write Hello World:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException
    {
        // Do required initialization
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter(); out.println("<html><body><b>"
        + message + "</b></body></html>"); out.close();

    }

    public void destroy() { }
}
```

Output:

finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:



Parameter data and Query Strings

Servlet has methods to access data contained in HTTP Request (URL) sent to the server from the browser. The Query String portion of the HTTP request is so called parameter data. For example,

`http://www.example.com/servlet/PrintThis?name=Raj&color=Red`

where the portion after the ? is called a query string. Here it is “name=Raj&color=Red”, in which name and color are parameter names and “Raj” and “Red” are parameter values. Printthis is a servlet filename and servelt is a directory. Multiple parameters are separated by &. All parameter values are strings by default. Parameter names and values can be any 8-bit characters.

The following methods are used to process these parameter data in sevlets.

TABLE 6.1: Some `HttpServletRequest` methods for accessing parameter data.

Method	Purpose
<code>String getQueryString()</code>	Returns the entire query string in its original (URL encoded) form.
<code>Enumeration getParameterNames()</code>	Returns Enumeration of String values representing all parameter names (URL decoded) in the query string.
<code>String getParameter (String name)</code>	Returns String representing value (URL decoded) of parameter named name, or null if parameter is not present in the query string.
<code>String[] getParameterValues (String name)</code>	Returns array of String's representing all values (URL decoded) of parameter named name, or null if parameter is not present in the query string.

The following program explains how to process these parameter names and values as well as path of the resource using servlet.

Example program

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NewServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {

            out.println("<html>");
            out.println("<head>"); out.println("<title>Servlet
NewServlet</title>"); out.println("</head>");

            out.println("<body>");
            out.println("Servlet file NewServlet is at: " + request.getContextPath());
            Enumeration para1=request.getParameterNames();
            while(para1.hasMoreElements())
            {
                out.println("Parameter name:"+para1.nextElement());
            }
            String name = request.getParameter("name");
```

```

        String id = request.getParameter("id");
        out.println("Name:" + name);
        out.println("Id:" + id);
        out.println("</body>");
        out.println("</html>");

    } catch (Exception e) {}
}
}

```

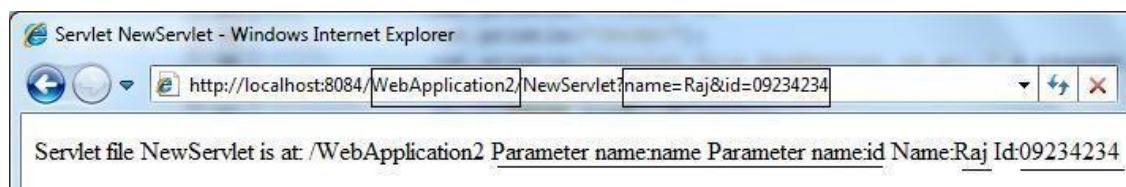
The method `getContextPath()` of `HttpServletRequest` object is used to get the location of the resource. The method `getParameter()` is used to get the value of the parameter. The method `getParameterNames()` is used to return the parameter names as well. It returns enumeration. The following code in the above program is used to retrieve the parameter names from the enumeration.

```

Enumeration para1=request.getParameterNames();
while(para1.hasMoreElements())
{
    out.println("Parameter name:"+para1.nextElement());
}

```

Output:



Forms and Parameter data:(Passing values from HTML document to Servlet)

A form automatically generates a query string when submitted. The parameter name specified by value of name attributes of form controls. For example,

```
<input type="text" name="username" size="40" />
```

where username is the parameter name.

Parameter value can be the value of value attribute of any form control or it may be the value received from the user by the control at run time. For example,

```

<label>
    <input type="checkbox" name="boxgroup1" value="tall"/>tall
</label>

```

Value for checkbox
specified by value attribute

The following program explains how to send the data to server from a web page and the same how to receive it from the server.

Html for creating a web page


```

<html>
<head>
</head>
<body>
<pre>
<form action="NewServlet" method="post">
  First Name: <input type="text" name="t1" />
  Last Name: <input type="text" name="t2" />
  Age: <input type="text" name="t3" /> E-mail:
  <input type="text" name="t4" />
  <input type="submit" value="Submit" />
</form>
</pre>
</body>
</html>

```

Servlet for processing the data coming from this web page

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NewServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {

            out.println("<html>");
            out.println("<head>"); out.println("<title>Servlet
NewServlet</title>"); out.println("</head>");

            out.println("<body>");

            String s1 = request.getParameter("t1");
            String s2 = request.getParameter("t2");
            String s3 = request.getParameter("t3");
            String s4 = request.getParameter("t4");

            out.println("First Name:" + s1);
            out.println("Last Name:" + s2);
            out.println("Age:" + s3);
            out.println("E-mail:" + s4);

            out.println("</body>");
            out.println("</html>");

        } catch (Exception e) {}
    }
}

```

Output

WEB TECHNOLOGY



GET vs. POST method for forms:

GET:

- It is used to process the query string which is part of URL
- If the length of query string is limited it may be used.
- It is recommended when parameter data is not stored but used only to request information.

POST:

- It is used to process the query string as well as to store the data on server.
- If the Query string is sent as body of HTTP request, the post method will be used to retrieve.
- If the length of query string is unlimited, it can be used
- It is recommended if parameter data is intended to cause the server to update stored data
- Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates

Important note:

For the HTTP Session, Cookies, URL rewriting use our class notes. The soft copy for these topics will be given later.

UNIT IV

Representing Web Data: XML-Documents and Vocabularies-Versions and Declaration -
Namespaces JavaScript and XML: Ajax-DOM based XML processing Event-oriented Parsing:
SAX-Transforming XML Documents-Selecting XMLData: PATH-Template- based
Transformations: XSLT-Displaying XML Documents in Browsers-Case Study- Related
Technologies. Separating Programming and Presentation: JSPTechnology Introduction-JSP and
Servlets-Running JSP Applications Basic JSP-JavaBeansClasses and JSP-Tag Libraries and Files-
Support for the Model-View-Controller Paradigm-Case Study-Related Technologies.

Representing Web Data: XML

XML

XML stands for eXtensible Markup Language, developed by W3C in 1996. XML 1.0 was officially adopted as a W3C recommendation in 1998. XML was designed to carry data, not to display data. XML is designed to be self-descriptive. XML is a subset of SGML that can define your own tags. A Meta Language and tags describe the content. XML Supports CSS, XSL, DOM.

The Difference between XML and HTML

1. HTML is about displaying information, where asXML is about carrying information. In other words, XML was created to structure, store, and transport information. HTML was designed to display the data.
2. Using XML, we can create own tags where as in HTML it is not possible instead it offers several built in tags.
3. XML is platform independent neutral and language independent.
4. XML tags and attribute names arecasesensitive where as inHTML it is not.
5. XML attribute values must be single or double quoted where as in HTML it is not compulsory
6. XML elements must be properlynested
7. All XML elements must have a closing tag
8. XML is used to create new internet languages. Here are some examples:
 - WSDL for describing available web services
 - WAP and WML as markup languages for handheld devices
 - RSS languages for news feeds
 - RDF and OWL for describing resources and ontology
 - SMIL for describing multimedia for the web

Well Formed XML Documents

XML with correct syntax is "Well Formed" XML. XML validated against a DTD is "Valid" XML.

A "Well Formed" XML document must have the following correct XML syntax:

- 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

Example for XML Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me! </body>
</note>
```

Xml document begins with XML declaration statement: `<? xml version="1.0" encoding="ISO-8859-1"?>` . The next line describes the root element of the document: `<note>`. This element is "the parent" of all other elements. The next 4 lines describe 4 child elements of the root: to, from, heading, and body. And finally the last line defines the end of the root element : `< /note>`

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...

XML vocabulary

XML vocabulary is used to define

-
- element and attribute names
 - element content
 - Semantics of elements and attributes

Some of the xml vocabularies are XHTML, RSS, XSL, DTD, and Schema

XML DTD

Document Type Definition purpose is to define the structure of an XML document. It defines the structure with a list of defined elements in the xml document.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)> <!ELEMENT
from (#PCDATA)> <!ELEMENT heading
(#PCDATA)> <!ELEMENT body
(#PCDATA)>
]>
```

Where PCDATA refers parsed character data. In the above xml document the elements to, from, heading, body carries some text, so that, these elements are declared to carry text in DTD file.

This definition file is stored with .dtd extension.

XML Schema

It is an alternative to DTD to define the structure of an XML document.

```
<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>
```

This definition file is stored with .xsl extension.

XML DTD vs XML Schema

The schema has more advantages over DTD. A DTD can have two types of data in it, namely the CDATA and the PCDATA. The CDATA is not parsed by the parser whereas the PCDATA is parsed. In a schema you can have primitive data types and custom data types like you have used in programming.

XML Parsers

An XML parser converts an XML document into an XML DOM object - which can then be manipulated with a JavaScript.

Two types of XML parsers:

- Validating Parser
 - It requires document type declaration
 - It generates error if document does not
 - Conform with DTD and
 - Meet XML validity constraints
- Non-validating Parser
 - It checks well-formedness for xml document
 - It can ignore external DTD

XML Namespaces

It is a collection of element and attributes names associated with an XML vocabulary. XML Namespaces provide a method to avoid element name conflicts.

XML document 1

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

XML document2

```
<table>
  <name> 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. Such name conflicts in XML can easily be avoided using a name prefix as shown below:

```
<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>
```

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"`

For example,

```
<h:table xmlns:h="http://www.w3.org/table">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="http://www.w3.org/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width> <f:length>120</f:length>
</f:table>
```

Default namespace

Default xml namespace uri for elements of a document is

`xmlns="http://www.w3.org/1999/xhtml"`

RSS

RSS stands for Rich Site Summary

RSS is a format for delivering regularly changing web content. Many news-related sites and other online publishers syndicate their content as an RSS Feed to whoever wants it.

- RSS allows you to syndicate your site content
- RSS defines an easy way to share and view headlines and content
- RSS files can be automatically updated
- RSS allows personalized views for different sites
- RSS is written in XML

With RSS it is possible to distribute up-to-date web content from one web site to thousands of other web sites around the world.

RSS allows fast browsing for news and updates. RSS was designed to show selected data.

Distributing your content using RSS will involve creating one file that contains your content. This file will reside on your server to enable other web sites to display your channel. You can update your channel simply by updating your file.

Without RSS, users will have to check your site daily for new updates. This may be too time-consuming for many users. With an RSS feed, they can check your site faster using an RSS aggregator (a site or program that gathers and sorts out RSS feeds) since RSS data is small and fast-loading

RSS is useful for web sites that are updated frequently, like:

1. News sites - Lists news with title, date and descriptions
2. Companies - Lists news and new products
3. Calendars - Lists upcoming events and important days
4. Site changes - Lists changed pages or new pages

Creating an RSS File

Your first step will be to identify your file. To do this, place the following code at the top of your text file.

```
<?xml version="1.0"?>
<rss version="0.91">
```

Your next step will be to create your channel header. The "channel" tag indicates that you are beginning a new channel.

```
<channel>
<title>Web-Source.net Syndication</title>
<link>http://www.web-source.net</link>
<description>Web Development article syndication feeds!</description>
<language>en-us</language>
```

The "title" tag indicates the name of your channel. The "link" tag will contain a link to your web site. The "description" tag describes your channel and the "language" tag indicates that you're writing in US English.

Now, you're ready to create your headlines. Each new "item" tag represents a new topic.

```
<item>
<title> Creating A Customized Marquee </title>
<link>http://www.example.com/tips.htm</link>
<description>
    Learn how to create a customized marquee for your web
</description>
</item>
```

Your final step will be to close your channel by adding the following tags:

```
</channel>
</rss>
```

Save your new file with .rss file extension and upload it to your server. And now, you're ready to share your content.

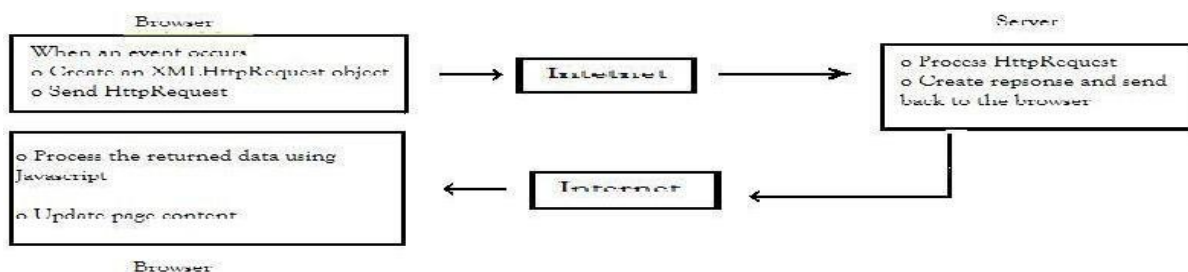
JavaScript and XML: AJAX

AJAX:

- AJAX stands for Asynchronous JavaScript andXML
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Using Ajax,
 - o Update a web page with new data without reloading the page
 - o Request data from a server after the page has loaded
 - o Receive data from a server after the page has loaded

- Send data to a server in the background
 - AJAX allows updating parts of a web page, without reloading the whole page.
 - With Ajax, web applications can also retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
 - Ajax is combination of
 - HTML
 - XML
 - CSS
 - JavaScript
 - JavaScript DOM
 - XMLHttpRequest in asynchronous mode
1. Asynchronous
 - This means that when you send a request, you wait for the response to come back, but are free to do other things while you wait. The response probably won't come back immediately, so you set up a function that will wait for the response to be sent back by the server, and react to it once that happens.
 2. JavaScript
 - JavaScript is used to make a request to the server. Once the response is returned by the server, you will generally use some more JavaScript to modify the current page's document object model in some way to show the user that the submission went through successfully.
 3. XML
 - The data that you receive back from the server will often be packaged up as a snippet of XML, so that it can be easily processed with JavaScript. This data can be anything you want, and as long as you want.

How AJAX works?



XMLHttpRequest - Constructors for other host objects allow a JavaScript program to send an HTTP request to a server and receive back a response containing an XML document. The following program illustrates the use of XMLHttpRequest.

Example

The following AJAX application above contains one div section and one button.

The div section will be used to display information returned from a server. The button calls a function named loadXMLDoc(), if it is clicked:

```
<html>
<head>
<script type="text/javascript">
function dataData()
{
    var xmlhttp=new XMLHttpRequest(); // create XMLHttpRequest to exchange data with server
    var obj= getElementById("myDiv");
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            obj=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","data.txt", true);
    xmlhttp.send();
}
</script>
</head>
<body><form>

<div id="myDiv">Let AJAX change this text</div>
<input type="button" value="Change Content" onclick="getData()" />

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

} To send a request to server

The AJAX application above contains one div section and one button. The div section will be used to display information returned from a server. Button calls a function named getData(), if it is clicked. The script section contains the getData() function. The AJAX script presents inside the getData() function. When a request is sent to server, onreadystatechange event is triggered. The readystate property holds the status of the XMLHttpRequest. The readystate=4 means request is finished and response is ready where as status=200 means "OK".

When the button is clicked, the content available in the file "data.txt" will be placed inside the <div> tag. The existing content "Let AJAX change this text" will be replaced with data.txt content

XML DOM

Document Object Model is for defining the standard for accessing and manipulating XML documents. XML DOM is used for

- Loading the xml document
- Accessing the xml document
- Deleting the elements of xml document
- Changing the elements of xml document

According to the DOM, everything in an XML document is a **node**. It considers

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

DOM Levels

- Level 1 Core: W3C Recommendation, October 1998
 - It has feature for primitive navigation and manipulation of XML trees
 - other Level 1 features are: All HTML features
- Level 2 Core: W3C Recommendation, November 2000
 - It adds Namespace support and minor new features
 - other Level 2 features are: Events, Views, Style, Traversal and Range
- Level 3 Core: W3C Working Draft, April 2002
 - It supports: Schemas, XPath, XSL, XSLT

We can access and parse the XML document in two ways:

- Parsing using DOM (tree based)
- Parsing using SAX (Event based)

Parsing the XML doc. using DOM methods and properties are called as tree based approach whereas using SAX (Simple Api for Xml) methods and properties are called as event based approach.

DOM based XML Parsing:(tree based)

JAXP is a tool, stands for Java Api for Xml Processing, used for accessing and manipulating xml document in a tree based manner.

In this approach, to access XML document, the document object model implementation is defined in the following packages:

- javax.xml.parsers
- org.w3c.dom

The following DOM javaClasses are necessary to process the XML document:

- **DocumentBuilderFactory** class creates the instance of DocumentBuilder.
- **DocumentBuilder** produces a Document (a DOM) that conforms to the DOM specification

The following methods and properties are necessary to process the XML document:

Property	Meaning
nodeName	Finding the name of the node
nodeValue	Obtaining value of the node
parentNode	To get parnet node
childNodes	Obtain child nodes
Attributes	For getting the attributes values

Method	Meaning
getElementByTagName(name)	To access the element by specifying its name
appendChild(node)	To insert a child node
removeChild(node)	To remove existing child node

To Count the Elements in a XML File

This program takes a file name from the console and checks its availability. If the file exists then a parser is created using Document Builder. This object parses the givenXML document. It searches thespecified element name and counts its occurrence in the xml file. If the given element doesn't exist it displays the '0' element.

Step 1: Create an xml file: Employee-Detail.xml

```
<?xml version = "1.0" ?>
<Employee-Detail>
<Employee>
    <Emp_Id> E-001 </Emp_Id>
    <Emp_Name> Vinod </Emp_Name>
    <Emp_E-mail> Vinod@yahoo.com </Emp_E-mail>
</Employee>
<Employee>
    <Emp_Id> E-002 </Emp_Id>
    <Emp_Name> Arun </Emp_Name>
```

```

    <Emp_E-mail> Arun@yahoo.com </Emp_E-mail>
</Employee>
</Employee-Detail>

```

Step 2: Create a java based dom for counting the number of elements in xml file.

```

import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;

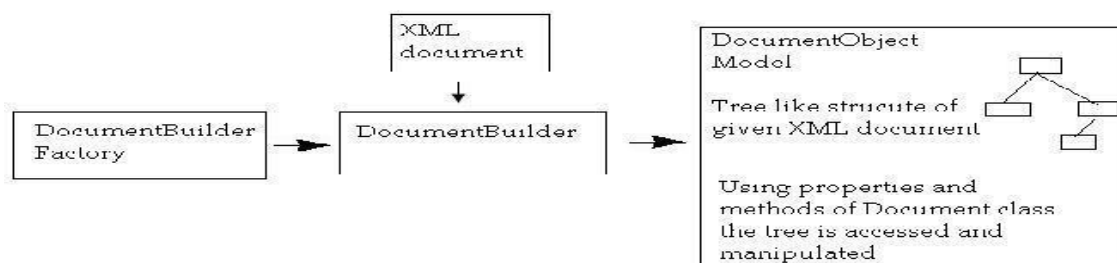
public class CountNodes
{
    public static void main(String[] args)
    {
        try
        {
            File file= new File(args[0]); // Employee_Detail.xml is a file to be parsed is given as
                                           //input from command prompt
            DocumentBuilderFactory factory= DocumentBuilderFactory.newInstance();
            DocumentBuilder builder= factory.newDocumentBuilder(); // parser is created
            Document doc= builder.parse(file); // file is parsed
            NodeList list = doc.getElementsByTagName("Emp_Id"); // the element to be searched is
                                                                specified
            System.out.println("Number of nodes: " + list.getLength()); // counting no. of occurrences
        } catch (Exception e){ }
    }
}

```

Output:

Number of nodes: 2

The diagram here shows the **JAXP APIs** to process xml document using the **DOM parser**:



Event oriented XML Parsing: SAX

SAX stands for Simple API for XML

SAX provides a mechanism for reading data from an XML document, SAX parsers operate on each piece of the XML document sequentially.

It is a kind of event oriented approach for parsing the xml document.

An XML tree is not viewed as a data structure, but as a stream of **events** generated by the parser.

The kinds of events are:

- the **start** of the document is encountered
- the **end** of the document is encountered
- the **start tag** of an element is encountered
- the **end tag** of an element is encountered
- **character data** is encountered
- a **processing instruction** is encountered

Scanning the XML file from start to end, each event invokes a corresponding **callback** method that the programmer writes.

SAX packages

- `javax.xml.parsers`: Describing the main classes needed for parsing
- `org.xml.sax`: Describing few interfaces for parsing

SAX classes

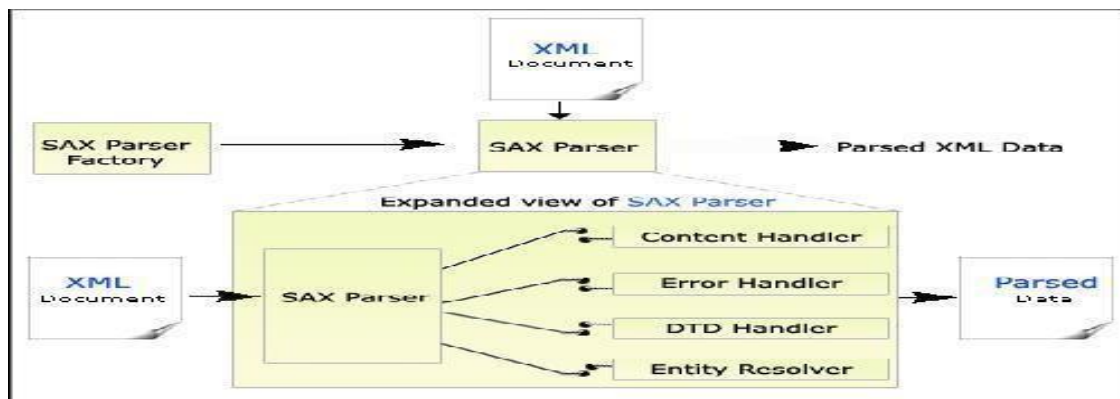
- **SAXParser** Defines the API that wraps an XMLReader implementation class
- **SAXParserFactory** Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents
- **ContentHandler** Receive notification of the logical content of a document.
- **DTDHandler** Receive notification of basic DTD-related events.
- **EntityResolver** Basic interface for resolving entities.
- **ErrorHandler** Basic interface for SAX error handlers.
- **DefaultHandler** Default base class for SAX event handlers.

SAX parser methods

- **startDocument()** and **endDocument()** – methods called at the start and end of an XML document.
- **startElement()** and **endElement()** – methods called at the start and end of a document element.

- **characters()** – method called with the text contents in between the start and end tags of an XML document element.

Understanding SAX Parser



At the very first, create an instance of the **SAXParserFactory** class which generates an instance of the parser. This parser wraps a SAXReader object. When the parser's **parse()** method is invoked, the reader invokes one of the several callback methods (implemented in the application). These callback methods are defined by the interfaces **ContentHandler**, **ErrorHandler**, **DTDHandler**, and **EntityResolver**.

SAX Example :Selecting XML data

Step 1. Create a XML file

Create a simple XML file as following

```
<?xml version="1.0"?>
<company>
  <staff>
    <firstname>Ram</firstname>
    <salary>100000</salary>
  </staff>
  <staff>
    <firstname>Kumar</firstname>
    <salary>200000</salary>
  </staff>
</company>
```


2. Create a Java file.

Use SAX parser to parse the XML file.

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ReadXMLFileSAX
{
    public static void main(String argv[])
    {
        try{
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();

            DefaultHandler handler = new DefaultHandler() {

                boolean bfname = false;
                boolean bsalary = false;

                public void startElement(String uri, String localName, String qName, Attributes
                                         attributes)
                    throws SAXException
                {
                    System.out.println("Start Element : " + qName);
                    if (qName.equals("FIRSTNAME")) { bfname = true; }
                    if (qName.equals("SALARY")) { bsalary = true; }
                }

                public void endElement(String uri, String localName, String qName) throws
                                         SAXException
                {
                }

                public void characters(char ch[], int start, int length) throws SAXException
                {
                    if(bfname)
                    {
                        System.out.println("First Name : " + new String(ch, start, length));
                    }
                }
            };
        }
    }
}
```

```

        bfname = false;
    }

    if (bsalary)
    {
        System.out.println("Salary : " + new String(ch, start, length));
        bsalary = false;
    }
}
}
saxParser.parse("c:\\file.xml", handler);

} catch(Exception e) {}
}
}

```

Result

Start Element :company
 Start Element :staff
 Start Element :firstname
 First Name : Ram
 End Element :firstname
 Start Element :salary
 Salary : 100000
 End Element :salary
 End Element :staff
 Start Element :staff
 Start Element :firstname
 First Name : Kumar
 End Element :firstname
 Start Element :salary
 Salary : 200000
 End Element :salary
 End Element :staff
 End Element :company

Differences between SAX and DOM

SAX	DOM
Event based model	Tree data structure
Serial access	Random access
Low memory usage	High memory usage
Used to process parts of the document	Used to edit the document

Used to process the document only once	Used to process multiple times (document is loaded in memory)
Parses node by node	Stores the entire XML document into memory before processing
Doesn't store the XML in memory	Occupies more memory
We can't insert or delete a node	We can insert or delete nodes
Top to bottom traversing	Traverse in any direction.
SAX is a Simple API for XML	Document Object Model (DOM) API
Packages required to import import javax.xml.parsers.*; import org.xml.sax.*;	import javax.xml.parsers.*; import javax.xml.parsers.DocumentBuilder; import javax.xml.parsers.DocumentBuilderFactory;
SAX generally runs a little faster than DOM	DOM is slow rather than SAX

Transforming XML document

It is process of extracting one info. From one xml doc. and uses that info to create another xml doc.

XSL

XSL is stands for eXtensible Style sheet Language which is an xml vocabulary. It contains two types of information:

Template data: Which is text that is copied to output xml document with change or no change.

XSL markup: which controls transformation process.

It uses two namespaces:

- <http://www.w3.org/1999/xsl/Transform> is the namespace name for xsl namespace
- <http://www.w3.org/1999/xhtml> is the xhtml namesapce name

XSL components: Three components

1. XSLT
2. XPATH
3. XSL-FO

XSLT – XSL Transformations (XSLT) is a language to specify transformations of XML documents. A transformation expressed in XSLT describes rules for transforming a source tree into a result tree

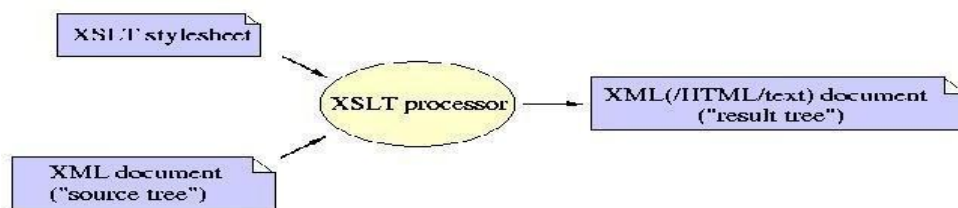
XPATH- is used to find information in an XML document. It navigates through elements and attributes in XML documents.

XSL-FO is a XSL Formatter an xml vocabulary for defining style properties of xml document.

To transform xml document the following things are required:

- a. Source xml document
- b. Xslt style sheet that defines the information of source xml program to be transformed
- c. Xslt processor that performs transformation

The following program illustrates that.



The JAXP is a tool which is used as XSLT processor. The XSLT processor receives xslt and xml program as its input and generates another xml program as its output where we can see the content retrieved from source xml program

The JAXPAPI has in the following packagesfor xml transformations:

Package	Description
javax.xml.transform	Defines theTransformerFactoryandTransformer classes. These classes are used to get a object for doing transformations.
javax.xml.transform.dom	Defines classes used to create input and output objects from a DOM.
javax.xml.transform.sax	Defines classes used to create input from a SAX parser and output objects from a SAX event handler.

javax.xml.transform.stream	Defines classes used to create input and output objects from an I/O stream.
----------------------------	---

Example for XSLT

You can refer lab experiment also

Step 1. Create an XML file and save as

The code for the emp.xml file is given below:

```
<?xml version = "1.0" ? >
<Employee-Detail>
  <Employee>
    <Emp_Id> E-001 </Emp_Id>
    <Emp_Name> Nisha </Emp_Name>
    <Emp_E-mail> Nisha1@yahoo.com </Emp_E-mail>
  </Employee>
  <Employee>
    <Emp_Id> E-002 </Emp_Id>
    <Emp_Name> Amit</Emp_Name>
    <Emp_E-mail> Amit2@yahoo.com </Emp_E-mail>
  </Employee>
</Employee-Detail>
```

Step 2: Create an XSLT Stylesheet and save as myXSL.xml extension

```
<?xml version="1.0" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html><head><title> XSL for Transforamtion </title>
    <body><p>
      <xsl:for-each select="Employee"> <xsl:value-of
        select="child::Emp_Id" />
        <xsl:value-of select="child::Emp_Name" />
        <xsl:value-of select="child::Emp_E-mail" />
      </xsl:for-each>
    </p></body></html>
  </xsl:template>
</xsl:stylesheet>
```

Step 3: Create XML Transformer program in java and give input both .xml and .xsl file

```

import javax.xml.transform.Transformer; import
javax.xml.transform.TransformerFactory; import
javax.xml.transform.stream.StreamSource; import
javax.xml.transform.stream.StreamResult; import
javax.xml.transform.Source;
import javax.xml.transform.Result;
import javax.xml.transform.OutputKeys;

```

```

public class XMLwithXSLT
{
    public static void main(String[] args) throws Exception
    {
        Source source = new StreamSource("myXML.xml");
        Source xsl = new StreamSource("myXSL.xsl");

        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer transformer = factory.newTransformer(xsl);

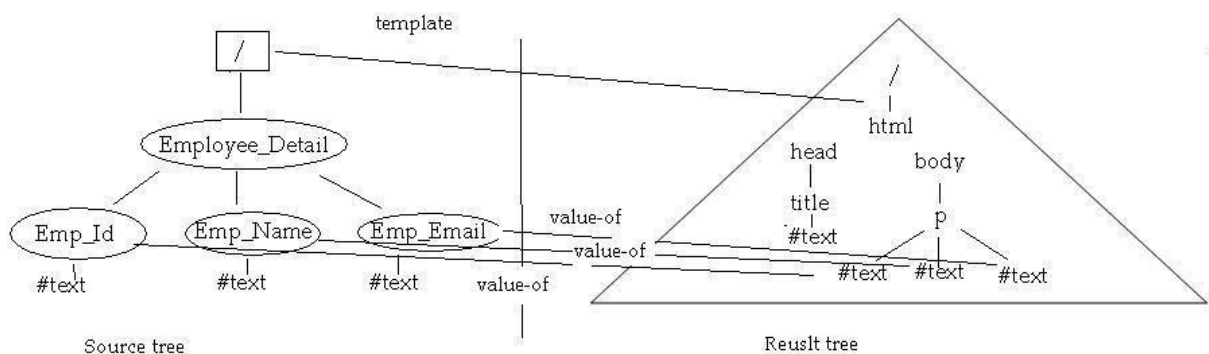
        new StreamResult(System.out);
    }
}

```

Output of the program:-

E-001	Nisha	Nisha1@yahoo.com
E-002	Amit	Amit2@yahoo.com

The following figure illustrates how the xslt transformer constructs the source tree from source xml document and by using that with the help of xsl program how the result tree is constructed.



Result tree is constructed based on the following definitions given in xsl style sheet

```

<xsl:template match="/">

```

```

<html><head><title> XSL for Transforamtion </title>
<body><p>
    <xsl:for-each select="Employee"> <xsl:value-of
        select="child::Emp_Id"/>
        <xsl:value-of select="child::Emp_Name"/>
        <xsl:value-of select="child::Emp_E-mail"/>
    </xsl:for-each>
</p></body></html>
</xsl:template>

```

XPATH

Is the syntax for specifying a collection of elements on other info contained within xml doc. XPATH expression will be applied to parse the xml doc. which constructs a tree like representation of xml doc. similar to DOM tree.

The root of the XPath parse tree is „document“

Location Paths

<xsl:template match="/"> which represents the XPath document root. An XPath expression such as this represents one or more nodes within XPath parse tree is known as location path.

This location path consists a „location step“

Location step has two parts:

- An axis name followed by (::)
- Node test

child::Emp_Id is an example of a location step and has two parts:

child :: **Empid**
Axis name *Node test*

Axis name specifies the direction to which we can search a node

Node test specifies an element name selected for transformation

Axis name:

The following are several axis names uses to search a particular element in the xml doc.

Name	Relationship with context node
self	The context node itself
child	Any immediate descendant
descendant	Any proper descendant
descendant-or-self	Any descendant, including the context node itself
parent	Immediate ancestor
ancestor	Any proper ancestor, including the document root (unless the context node is the document root)
ancestor-or-self	Any ancestor, including the context node itself
preceding-sibling	Any sibling of the context node that precedes the context node in the document
following-sibling	Any sibling of the context node that follows the context node in the document
attribute	Any attribute defined for the context node

Predicate:

XPath uses some predicate which constructs the node test. Predicate can be either

child::<element name>

(Or)

Attribute::<value>

Absolute and Relative Location Path

A location path can consist of several location steps separated by / character.

For example,

Relative location path

```
<xsl:value-of select="child::Emp_Id/child::Emp_Name"/>
```

Absolute location path

```
<xsl:value-of select="/child::Emp_Id/child::Emp_Name"/>
```

Displaying XML in the browser: (XML+CSS program)

XML uses a processing instruction which is a special xml tag that has syntax similar to that of xml declaration. The tag begins with <? and ends with ?>. Each processing instruction begins with the name as the target of the processing instruction. The tag "xml-stylesheet" is used to link the xml document with the css file.

HelloWorld.xml

```
<? xml version="1.0" encoding="UTF-8" ?>
<? xml-stylesheet type="text/html" href="helloworld.css" ?>
<message> Hello World! </message>
```

Helloworld.css

```
message {
    display:block;
    margin: 8pt; font-
    weight=bold;
}
```

When xml doc. Helloworld.xml is executed, the xml doc is linked to the specified "Helloworld.css" file and displays the output after applying css specifications.

Separating Programming and Presentation: JSP Technology

What is JSP?

- JSP Stands for "Java Server Pages". Using "JSP" we can use both, static HTML with dynamically-generated HTML. Web pages created using CGI programs are mostly static , dynamic part is limited to a few small locations. But using CGI and servlet , you can generate entire page through one program. Using JSP, you can built two parts separately.

JSP is the product of Sun Microsystems Inc.JSP has more advance features than Servlet . JSP separates the presentation logic from the business logic ,provide facility to developers to work separately without any trouble.JSP have the properties of Cold fusion and ASP and hence provide the flexibility to embed the business logic efficiently within the HTML content (presentation logic).

Advantages of JSP:-

- JSP is useful for server side programming
- JSP are translated and compiled into JAVA servlets but are easier to develop than JAVA servlets.
- JSP uses simplified scripting language based syntax for embedding HTML into JSP.
- JSP containers provide easy way for accessing standard objects and actions.
- JSP reaps all the benefits provided by JAVA servlets and web container environment, but they have an added advantage of being simpler and more natural program for web enabling enterprise developer
- JSP use HTTP as default request /response communication paradigm and thus make JSP ideal as Web Enabling Technology.

JSP and Servlets

- JSP documents are not executed directly
 - When a JSP document is first visited,
 1. first the server translates the JSP document to a servlet
 2. Compiles the servlet
 - Then the servlet is executed
 - If any error occurs while executing, the exception will occur which gives the information for the servlet not for the JSP
 - A JSP-generated servlet has a `_jspService()` method rather than `doGet()` or `doPost()`
 - This method begins to access the java code by creating a number of implicit objects.

Implicit Objects in JSP

JSP Implicit Objects are:

- Pre-defined variables that can be included in JSP expressions and scriptlets.
- Implemented from servlet classes and interfaces.

The followings are the implicit objects supported by jsp.

OBJECT	CLASS
Out	JSP writer
Request	HttpServletRequest
Response	HttpServletResponse
Session	HttpSession
Application	ServletContext
Config	Servlet Config

Page	Object

Difference between servlet and JSP

- In servlets both the presentation and business logic are placed together whereas in jsp both are separated by defining by java beans.
- Servlet is a java code where HTML code can be embedded whereas in jsp, it is a kind of HTML where java code can be embedded by using special tags.
- JSP allows creating and using custom tag libraries.

For example,

```
<%@page language="java" contentType="text/html" %>           // jsp element
<html><body>
    <% out.println("JSP = HTML+JAVA") %>                       // jsp element
    Today date is:
    <% = new Date().toString() %> // jsp element and java method
</body> </html>
```

In this code, jsp elements that use java code are merged with HTML code.

Scoped Variables

Four different types of variable scopes within the JSP page can be used:

- **Page Scope:** Objects stored in page scope can only be retrieved during the processing of that page for a specific request.
- **Request Scope:** Objects stored in request scope can be retrieved during the processing of all pages taking part in the processing of a request
- **Session Scope:** Object stored in session scope can be retrieved by any pages accessed by a user during a single interactive session with the Web application

-
- **Application Scope:** Object stored in application scope is accessible from all pages and for all users, until the Web application itself is unloaded

Display "Hello JSP" using Jsp expression

In this section, We will simply output "Hello JSP" in a web browser using Jsp expression. JSP expressions insert values directly into the output. The syntax to include a JSP expressions in the JSP file is:

```
<%= expression %>.
```

In JSPpage, we can use both, static HTML with dynamically-generated HTML. In this program, we produce output "hello Jsp" using this tag :

```
<%= "Hello World!" %>
```

All the other content except this is simple HTML code.

Code of this program:

```
<html>
    <head><title>Hello World JSP Page.</title></head>
    <body>
        <<%= "Hello JSP!" %>
    </body>
</html>
```

Output : Hello JSP!

Web Applications

- A web application is a collection of resources that are used together to implement some web-based functionality
- To develop a web applications the following resources are needed:
 - Server side Programming
 - Servlets /JSP,

-
- Client side programming : HTML documents, style sheets, XML , images, non servlet java classes etc,
 - Web form validation: JavaScript, non-servlet Java classes, *etc.*
 - Other resources:
 - Databases: MS-Access, MySQL/SQL
 - Web server: Apache Tomcat/ Java Web server/ Glass fish,
 - Browser: IE/Netscape/Chrome/FireFox

Running JSP Applications

The following section explains the steps you must take to run sample JSP into TomCat

The steps involved

1. Preparation
2. Deploy
3. Run

1. Preparation

Step 1: Create required jsp files like ExcelCalculatorJSP.html, and ExcelCalculatorExample.jsp files

Step 2. Create a folder under webapp directory in tomcat installation path
eg: C:\Tomcat5.5\webapps\test

Step 3. Copy all your JSP files in a folder by name "test".

2. Deploy

1. In the test directory, create two directories named META-INF and WEB-INF. Now you should have a directory structure that looks like this:



2. Create the XML descriptor: Here is the sample XML descriptor you can use to deploy this example. Save this file as web.xml in the WEB-INF directory.

```
<?xml version="1.0" encoding="Cp1252"?>
<web-app> <display-
  name>ExcelCalculatorJSP</display-name> <servlet>

  <servlet-name>ExcelCalculatorJSP </servlet-name>
  <display-name>ExcelCalculatorJSP</display-name>
  <jsp-file>/ExcelCalculatorExample.jsp </jsp-file>
</servlet>
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
</web-app>
```

3. Create a manifest file: Create a file called Manifest.mf (it can be an empty file) and save it in the META-INF directory.
4. After you finished putting these files on the correct place, your directory structure looks like this:



5. Now you need to add this new JSP to server.xml. Go to the conf directory and open server.xml.
6. Add the following lines at the bottom of server.xml

```
<Context path="/test" docBase="webapps/test" crossContext="true"
  debug="0" reloadable="true" trusted="false" >
</Context>
```

3. Run

1. Shut down the server by issuing this command: shutdown
2. Re-start server by issuing this command: startup
3. In browser, enter: <http://localhost:8080/test/ExcelCalculatorJSP.html>

Basic JSP

JSP page has the following components(contents)

- Directives

- Declarations
- Expressions
- Standard Actions
- Scriptlets

Directives

The directives are used to specify the attributes of the JSP page. Also it can be used to import java packages in to the jsp page. For example,

```
<%@ page language="java" contentType="text/html" %>
```

```
<% @page import ="java.util.*" %>
```

Where language="java" tells tha the page is contains java code, the code contentType="text/html" specifies the MIME type, the code import ="java.util.*" used to import all the classes of java"s utility package into this page.

Declarations: Provide a mechanism to define variables and methods. Declarative statements are placed within <%! and %> symbols and always end with a semicolon.

Syntax: <%! Declare all the variables here %>

For example, <% int a, b %>

Scriptlets: Consists of valid Java code snippets that are enclosed within <% and %> symbols. The syntax to declare JSP scriptlets to include valid Java code is:

```
<% Java code %>
```

Scriptlets can't generate HTML itself. Using "out" variable, it can generate HTML. This variable does not need to be declared. It is already predefined for Scriptlets, along with some other variables. The "out" variable is of type "**javax.servlet.jsp.JspWriter**".

For example,

```
<HTML>
<BODY>
<%
    java.util.Date date = new java.util.Date();
%>
Hello! The time is now
<%
    out.println( String.valueOf( date ));
%>
```

```
</BODY>
</HTML>
```

Output: Hello! The time is now 12/12/2010 12:10:02

Expression:

Insert values directly into the output. The syntax to include a JSP expression in the JSP file is:
<%= expression%>.

For example, <%= "Hello World!" %>

Actions

The elements which are used to access the java bean in a jsp page is called jsp action elements. JSP defines the following action elements:

- Standard action and Custom action
- JSTL

Standard action

Tags	Description
------	-------------

<jsp:useBean>	used to include java bean object
<jsp:setProperty:	used to set the property value for java bean class
<jsp:getProperty>	used to get the property of java bean class
<jsp:include>	used to include response from servlet or jsp when request is being processed
<jsp:param>	for adding specific parameter into the jsp page
<jsp:forward>	used to forward current request
<jsp:plugin>	used to embed java applet
<jsp:attribute>	used to set the value fo action attribute
<jsp:elelemt>	used to generate XML elements dynamically

Java Bean classes and JSP

Java Beans are reusable components. They are used to separate Business logic from the Presentation logic. Internally, a bean is just an instance of a class.

JSP's provide three basic tags for working with Beans.

- **<jsp:useBean id=*“bean name”* class=*“bean class”* scope = *“page | request | session | application”* />**

bean name = the name that refers to the bean.

Bean class = name of the java class that defines the bean.

- **<jsp:setProperty name = *“id”* property = *“someProperty”* value = *“someValue”* />**

id = the name of the bean as specified in the useBean tag.

property = name of the property to be passed to the bean.

value = value of that particular property .

An variant for this tag is the property attribute can be replaced by an *“*”*. What this does is that it accepts all the form parameters and thus reduces the need for writing multiple setProperty tags.

The only consideration is that the form parameter names should be the same as that of the bean property names.

- `<jsp:getProperty name = "id" property = "someProperty" />`

Here the property is the name of the property whose value is to be obtained from the bean.

Step 1: Create a java bean

```
public class SampleBean
{
    private String firstName = "";
    private String lastName = "";

    public void setFirstName(String name)    {  firstName = name;  }

    public void setLastName(String name)    {  lastName = name;  }

    public String getFullName() { return firstName + " " + lastName; }
}
```

Step 2: Use bean inside the jsp file

```
<HTML>
<HEAD>
<TITLE>Example: Simple Java Bean</TITLE>
<jsp:useBean id="SampleBean" scope="page" class="SampleBean" />
</HEAD>
<BODY>

<!-- Set bean properties --%>
<jsp:setProperty name="SampleBean" property="FirstName" value="Robert" />
<jsp:setProperty name="SampleBean" property="LastName" value="John" />

<!-- Get bean properties --%>
<jsp:getProperty name="SampleBean" property="FullName" />

</BODY>
</HTML>
```

JSTL

JSTL: JSP Standard Tag Libraries is a collection of JSP custom tags

The goal of JSTL, is to help simplify Java Server Pages page authoring tasks. To achieve this goal, JSTL has provided custom tags for many common JSP page authoring tasks that require scripting statements to manipulate server side dynamic data.

JSTL offers tags through 4 functional areas:

- core - Basic scripting
- xml - XML processing
- fmt - Internationalization of formatting
- sql - Data base accessing
- functions – for using functions

The following table summarizes these functional areas along with the prefixes used in the jsp pages.

Area	Description	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
Formatting		fmt
	Message formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

Core tag

- <c:set>
- <c:remove>
- <c:forEach>
- <c:if>
- <c:choose>
- <c:when>

<c:set>

The <c:set> action provides a tag-based mechanism for creating and setting scoped variables. For example,

```
<c:set var="variable_name" scope="page | request | session | application"
value="expression"/>
```

<c:remove>

The <c:remove> action is used to delete a scoped variable. For example,

```
<c:remove var="variable_name" scope="page | request | session | application" />
```

<c:forEach>

The <c:forEach> custom tag is used to fetch and display collections of data, typically in the form of a list or sequence of rows in a table. This tag supports two different styles of iteration:

```
<c:forEach var="variable_name" varStatus="name" begin="expression" end="expression"
step="expression">
    -----
</c:forEach>
```

The index of the iteration starts at the value of the **begin** attribute, is incremented by the value of the **step** attribute, and halts iteration when it exceeds the value of the **end** attribute. If the **step** attribute is omitted, the step size defaults to 1.

For example,

```
<c:forEach var="i" begin="10" end="20" step="2">
    <c:out value="${i}" />
</c:forEach>
```

Output: 10 12 14 16 18 20

<c:if>:

<c:if> evaluates a single test expression and then processes its body content only if that expression evaluates to true. If not, the tag's body content is ignored.

Syntax for the <c:if> conditional action:

```
<c:if test="expression" var="name" scope="scope">
    -----
</c:if>
```

For example,

```
<c:set var="fruit" />
<c:if test="${fruit=="Apple"}" />
I like <c:out value="${fruit}" /> very much!
</c:if>
```

```
<c:set var="fruit" />
<c:if test="${fruit=="Mango"}" />
I like <c:out value="${fruit}" /> very much!
</c:if>
```

<c:choose>:

<c:choose> is provided for cases in which mutually exclusive tests are required to determine what content should be displayed.

Syntax for the <c:choose> action:

```
<c:choose>
  <c:when test="expression">
    -----
  </c:when>
  -----
  <c:otherwise>
    -----
  </c:otherwise>
</c:choose>
```

For example,

```
<c:set var="fruit" />
<c:choose>
  <c:when test="${fruit=="Apple"}">
    I like <c:out value="${fruit}" /> much
  </c:when>
  <c:when test="${fruit=="Mango"}" >
    I like <c:out value="${fruit}" /> very much
  </c:when>
  <c:otherwise>
    I do not like <c:out value="${fruit}" />
  </c:otherwise>
```

</c:choose>

<c:out>

The <c:out> tag evaluates the expression specified by its value attribute, then prints the result. If the optional default attribute is specified, this action will instead print its value if the value attribute's expression evaluates either to null or an empty String.

<c:out value="expression" default="expression" />

Expression Language (EL)

This is one of the most important features of the JSTL and is a prominent feature of the JSP 2.0 specification.

The Expression Language or EL as it is known is used by JSP developers to access and use application data without using java code. In other words, *EL is the replacement for java code within the jsp page.*

The JSP 2.0 expression language helps to simplify the presentation layer by replacing hard-to-maintain Java scripting elements in jsp page.

The EL statements are always used within \${....}

Basic syntax

The syntax of expression language is very simple. EL expressions are invoked with this syntax: \${expr}, where expr represents any valid expression. For example,

$\${20 * 2}$

For example,

Step 1: create a html file to get a user name

```
<html> <body>
    <form method="post" action="ElDemo.jsp">
        Enter user name: <input type="text" name="t1" />
        <input type="submit" value="submit" />
    </form>
</body></html>
```

Step 2: Create a jsp file to obtain user name from the above html file using EL

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html><body>
    Entered user name is <c:out value="${param.t1}" />
</body></html>
```

EL Reserved Words:

Following words are EL keywords and hence must not be used as identifiers:

and eq gt true instance of mod
or ne le false empty not
lt ge null div

EL Operators

- EL includes several operators to manipulate and compare data accessed by EL expressions:

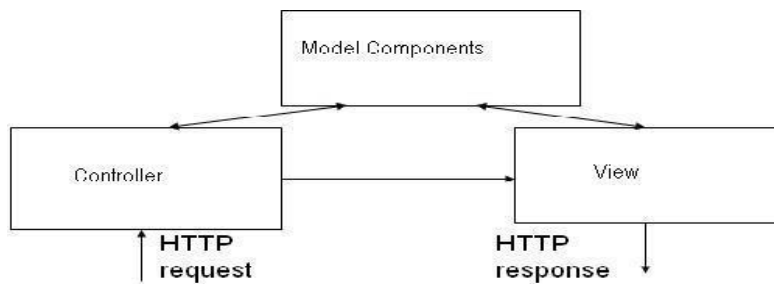
Category	Operators
Arithmetic	+, -, *, / or div, % or mod
Relational	== or eq, != or ne, < or lt, > or gt, <= or le, >= or ge
Logical	&& or and, or or, ! or not

MVC (Model -View - Controller)

Many web applications are based on the Model-View-Controller (MVC) architecture pattern. That is, web apps contains three parts: 1) business logic 2) presentation and 3) request processing

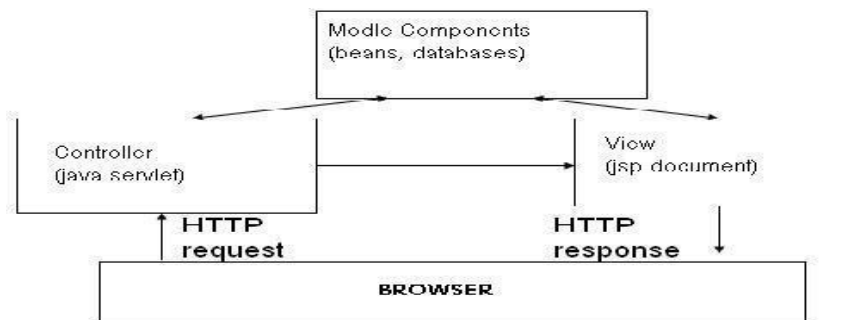
The key motivation behind the MVC approach is the desire to separate the code that creates and manipulates the data (i.e business logic) from the code that presents the data (i.e presentation) and the code that process the request (i.e controller)

Business logic means the code applied for manipulation of application data and presentation refers the code written for look and feel of the web page such as background color, font style, placing of form controls and so on. Controller means that process the request message.



According to MVC, the Model corresponds to business logic, the View corresponds to presentation and the Controller corresponds to request processing

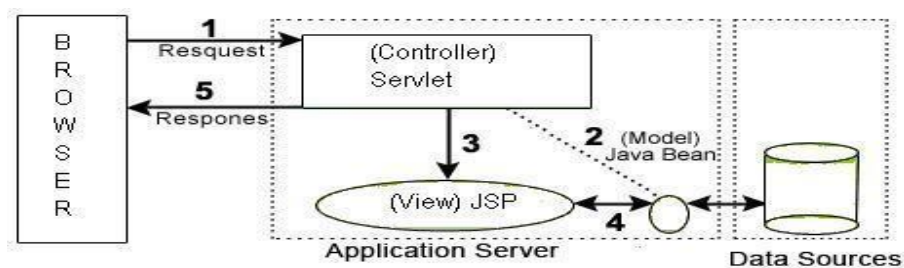
Typical JSP implementation of MVC



Java servlet program is the Controller component that receives the request from the browser.

Java bean class is the Model component that processes the business logic which can use the data source for that.

Jsp is the View component that presents the result of the business logic to the browser with the help of controller. The following program also illustrates that.



In this architecture, JSP is used for creating the view for the application. A centralized Servlet is used to handle the entire request for the application. The Servlet works as the controller for the application. It then uses the Java beans for processing the business logic and getting the data

(Model) from the database. Finally it uses the JSP to render the view which is displayed to the user.

Advantage of using MVC

- MVC allows the developer to keep the separation between business logic, presentation and request processing. Due to this separation, any changes to the presentation can be made easy without disturbing business logic.

UNIT V

Web Services: JAX-RPC-Concepts-Writing a Java Web Service-Writing a Java Web Service Client-Describing Web Services: WSDL- Representing Data Types: XML Schema-communicating Object Data: SOAP Related Technologies-Software Installation-Storing Java Objects as Files-Databases and Java Servlets

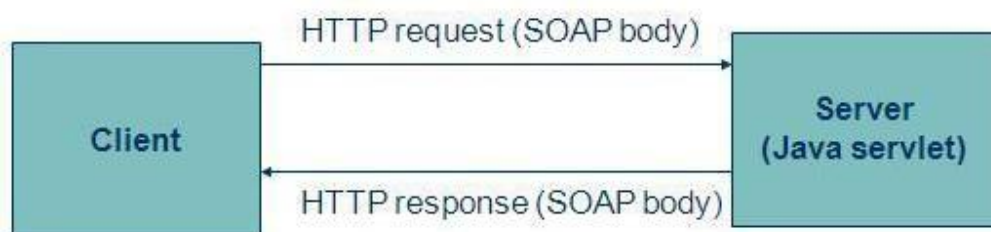
Web Application vs Web Services

- A web application uses Web technologies to provide functionality to an end user
- A web service uses Web technologies to provide functionality to another software application

Web Services

Web services are open standard based Web applications that interact with other web applications for the purpose of exchanging data.

XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, and then waits for a corresponding XML response. Because all communication is in XML, web services are platform neutral and language neutral. Java can talk with Perl; Windows applications can talk with UNIX applications. Web services conceptually are just specialized web applications:



Body of web services request and response will be SOAP message which defines a protocol for message exchange between applications.

Popular **examples** for Web Services

1. Currency Converter
2. Temperature conversion

3. Weather Forecast system
4. Credit card validation system

Standard web services technologies:

- SOAP
- WDSL
- XML Schema

Higher-level API's such as JAX-RPC and others are often used to automatically generate web services client and server communication software. Microsoft .NET framework is one popular alternative to JAX-RPC

JAX-RPC

It stands for Java API for XML-based RPC. JAX-RPC is a technology for building web services and clients that use *remote procedure calls* (RPC) and XML. Often used in a distributed client-server model, an RPC mechanism enables clients to execute procedures on other systems.

With JAX-RPC, clients and web services have a big advantage: the platform independence of the Java programming language. In addition, JAX-RPC is not restrictive: a JAX-RPC client can access a web service that is not running on the Java platform, and vice versa. This flexibility is possible because JAX-RPC uses technologies defined by the World Wide Web Consortium (W3C) such as:

- SOAP defines message format
- WSDL describes the various services offered by web service application as xml file
- XML Schema – defines data types used in WSDL doc.

Writing Web Service using JAX-RPC (Currency Converter)

These are the basic steps for creating the web service and client:

1. Create an interface
2. Code the implementation class.
3. Use `wscompile` to generate wsdl doc.
4. Create deployment descriptor file
5. Package the files into a WAR file.
6. Deploy the WAR file.
7. Create a client file to access the web service
8. Use `wscompile` to generate and compile the web service artifacts needed to connect to the service.

9. Compile the client class.
10. Run the client.

Example for Creating Web services software

To write web service using JAX-RPC, we need JWSDP (Java Web Server Development Package). This package has wscompile and wsdeploy tools. wscompile is used to convert java file into wsdl file whereas wsdeploy is used to package our web service.

To Create a web server file

Create a directory in the name of CurrencyConverter and create a sub director

myCurCon under WEB-INF/Src folder

Step 1: Create a service end point interface

Rules for creating service end point interface are as follows:

1. The interface must extend java.rmi.Remote interface
2. Every method in this interface must throw java.rmi.Remote exception
3. Every method return type and its parameter data type in this interface must be java primitive data types
4. The interface must not contain any public static final declarations.

```
package myCurCon; public
class ExchangeValues
{
    public double dollars;
    public double euros;
    public double yens;
}

package myCurCon;
public interface CurCon extends java.rmi.Remote
{
    public ExchangeValues fromDollars(double dollars) throws java.rmi.RemoteException;
    public ExchangeValues fromEuros(double dollars) throws java.rmi.RemoteException;
    public ExchangeValues fromYens(double dollars) throws java.rmi.RemoteException;
}
```

Step 2: Create a class to implement this interface

```

public class CurConImpl implements CurCon
{
    public ExchangeValues fromDollars(Double Dollars) throws java.rmi.RemoteException
    {
        ExchangeValues ev=new ExchangeValues();
        ev.dollars=dollars;
        ev.euros=dollars*100;
        ev.yens=dollars*200;
        return ev;
    }
}

```

Step 3: Compile using javac to create respective classes for ExchangeValues.java, CurCon.java and CurConImpl.java

Step 4: Use wscompile to create WSDL doc. Inputs to wscompile are supplied via the following configuration file called config.xml file.

```

<? xml version="1.0" ?>
<configuration xmlns="url" >
    <service name="HistoricCurrencyConverter" targetNameSpace="url"
        typeNameSpace="url" packageName="myCurCon">
        <interface name="myCurCon.CurCon" />
    </service>
</configuration>

```

- Save this file as config.xml and use the following command to create wsdl doc.

```

wscompile -define -d WEB-INF -classpath WEB-INF/classes -model WEB-INF/model.xml.gz
config.xml

```

-d specifies the directory to receive the generated wsdl doc. which will be named as Historic Currencyconverter.wsdl

Step 5: Create another configuration file called jaxrpc-ri.xml which is used as deployment descriptor file as shown below

```

<? xml version="1.0" ?>
<webServices xmlns="url" targetNameSpace="url"
    typeNameSpace="url"
    urlPatternBase="/converter" >
    <endpoint name="CurrConverter" displayName="Currency
        Converter" description="Converts b/w dollars, yens
        and euros" interface="myCurCon.curCon"
        model="/WEB-INF/model.xml.gz"
        implementation="myCurCon.CurConImpl"/>
    <endpointMapping endPointName="CurrConverter"

        urlPattern="/Currency" />
</webServices>

```

The tag <endpoint> gives the information to web server including name of the service end point interface, implementation file model generated by wscompile etc.

Step 6: Package and deploying the web service created

Create WAR(Web Archive) file. Add the deployment descriptor and service files to the WAR file. The following command is used:

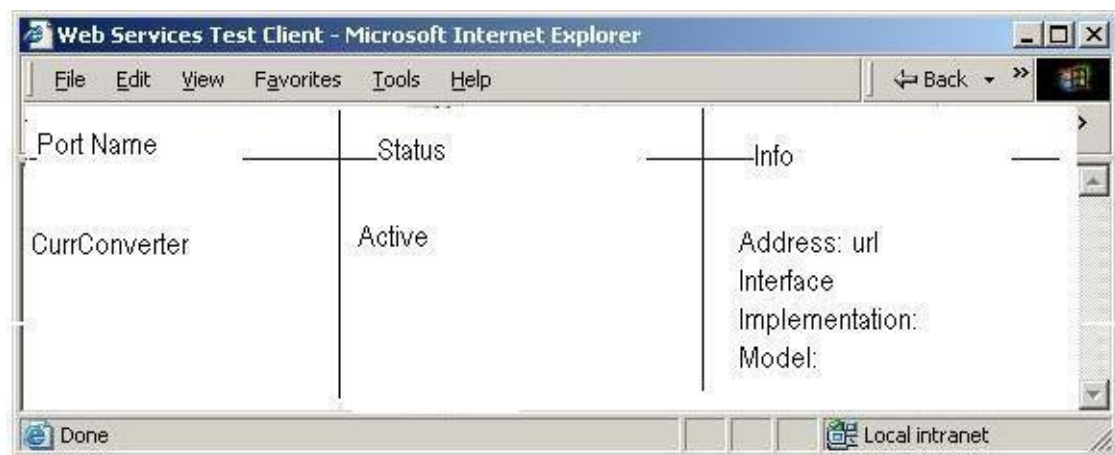
```
jar cf converter-temp.war WEN-INF
```

Deploy the war file using the following command

```
wsdeploy -o converter.war converter-temp.war
```

Step 7: Start tomcat server

Step 8: Browse to <http://localhost:8080/Converter/Currency> to see the status of the currency converter web service. The output will be:



To write a client program

To access the currency converter web service it is necessary to have a java client program

Step 1: Create xml file named config.xml

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

```
<configuration xmlns="url" wsdl location ="http://localhost:8080/Converter/Currency?wsdl"
```

```
    packageName="myCurConClient" />
```

```
</configuration
```

Step 2: Use wscompile to load wsdl from the url specified in the location attribute of config.xml. Use the following command for that

wscomplie -gen -keep -d classes -s src config.xml

This instructs to generate a class file from the wsdl and keep the java source file in src folder.

Step 3: Create java client program for accessing the web services as shown below

```
package mycurConClient;
public class CurConBean
{
    private double value=1.0; private
    String Currency="dollars";
    public void setValue(double value)
    {
        this.value=value;
        return;
    }
    public void setCurrency(String currency)
    {
        this.currency=currency;
        return;
    }
    public ExchangeValues getExValues()
    {
        ExchangeValues ev=null;
        CurCon=(new HistoricCurrencyConverter_impl().getCurConPort());
        try
        {
            if(currecny.equals("euros")
            {
                ev=curcon.fromEuros(value);
            }
            else if(currency.equals("yens")
            {
                ev=curcon.fromYens(value);
            }
        }catch(Exception e){}
        return ev;
    }
}
```

Step 4: Compileusing javac

Step 5: Create a jsp for using this bean class to perform a currency conversion class and display the result in a HTML file

```
<html>
<jsp:useBean id="client" class="myCurConClient.CurConBean" />
<body>
<%
```

```

        String value=request.getParameter(val);
    %>

    <jsp:setProperty name="client" property="Currency" value="curr" />

    <jsp:setProperty name="client" property="Currency" value="cur" />
    Converted amount:
    </jsp:getProperty name="Client" property=ExValue" />
    </body>
    </html>

```

Step 6: In the browser location bar type

http://localhost:8080/ConverterClient/Conver.jsp?cur="euros" val="500.30"

This invokes the web service from the server and converts the euros value 500.30 to equivalent dollars and yens value and displays the result.

SOAP

SOAP is a simple XML-based protocol that allows applications to exchange information over HTTP.

WSDL

Web Services Description Language is the standard format for describing a web service in XML format.

WSDL definition describes how to access a web service and what operations it will perform.

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Then the client can then use SOAP to actually call one of the functions listed in the WSDL.

The WSDL Document Structure

The main structure of a WSDL document looks like this:

```

<definitions>
<types>
    definition of types.....
</types>

```



```

<message>
  definition of a message....
</message>

<portType>
  <operation>
    definition of a operation.....
  </operation>
</portType>

<binding>
  definition of a binding....
</binding>

<service>
  definition of a service....
</service>

</definitions>

```

A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

A WSDL document describes a web service using these major elements:

Element	Defines
<types>	The data types used by the web service
<message>	The messages used by the web service
<portType>	The operations performed by the web service
<binding>	The communication protocols used by the web service

A WSDL document can also contain other elements, like extension elements, and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

Note: The second set of lines of the WSDL document is optional

First lines of the WSDL document

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

```

<definitions name=" " targetNamespace="uri"
              xmlns:tns="uri"
              xmlns="uri"
              xmlns:ns2="uri"
              xmlns:xsd="uri"
              xmlns:soap="uri" >

```

Second set of lines of the WSDL document

```

<types>
  <schema>
    <import namespace="uri" location="uri"/>
    <complexType name=" ">
      <sequence>
        <element name=" " type=" " />
        :
        :
      </sequence>
    </complexType>
  </schema>
</types>

```

The types element defines data type definitions that are relevant for the exchanged messages. The content of types is normally XML schema markup. The data type can be either complex type or simple type.

The sequence element is a collection of <element> tags. Each element tag specifies name of the operation (service or method) provided by the web service and its data type of the return value.

Third set of lines of the wsdl document

```

<message name=" " >
  <part name=" " type=" " />
</message>
:
:
<message name=" " >
  <part name=" " type=" " />
</message>

```

- The message name attribute provides a unique name among all messages defined within the enclosing WSDL document.
- Messages consist of one or more logical parts. Parts are a flexible mechanism for describing the logical abstract content of a message.
- Each part is associated with a type attribute. Type attribute value can be either complex type or simple type

Fourth set of lines of the wsdl document

```
<portType name=" ">
  <operation name=" " parameterOrder=" ">
    <input message=" " />
    <output message=" " />
  </operation>
  -----
  -----
  -----
</portType>
```

- A portType is a named set of abstract operations and the abstract messages involved.
- The portType name attribute provides a unique name among all port types defined within in the enclosing WSDL document.
- An operation is named via the nameattribute.
- The message attribute of the input and output elements provides a unique name among all input and output elements within the enclosing portType.
- parameterOrder attribute reflects the order of the parameters in the RPC signature

Fifth set of lines of wsdl document

```
<binding name=" " type=" ">
  <operation name=" ">
    <input>
      <soap:body encodingStyle="uri " use="encoded" namespace="uri" />
    </input>
    <output>
      <soap:body encodingStyle="uri " use="encoded" namespace="uri" />
    </output>
    <soap:operation soapAction=" " />
  </operation>
</soap:binding transport=uri" style="rpc" />
```

</binding>

- A binding defines message format and protocol details for operations and messages defined by a particular portType. The name attribute provides a unique name among all bindings defined within in the enclosing WSDL document. A binding references the portType that it binds using the type attribute
- An operation element within a binding specifies binding information for the operation with the same name within the binding's portType.
- The content of operation element in a binding is a pair of input and output elements.
- The soap:body element specifies how the message parts appear inside the SOAP Body element.
- The required use attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message. If use is encoded, then each message part references an abstract type using the type attribute. These abstract types are used to produce a concrete message by applying an encoding specified by the encodingStyleattribute.
- The soap:operation element provides information for the operation
- The styleattribute indicates whether the operation is RPC-oriented or not.
- The soapAction attribute specifies the value of the SOAPAction header for this operation

Final lines of wsdl document

```
<service name=" ">
  <port name=" " binding="internet address ">
    <soap:address location=" " />
  </port>
</service>
```

</definitions>

A service groups a set of related ports together. This provides a name for overall web service. Each port associates a binding with an internet address. This address is specified by including a soap: address element in the content of the port as shown.

WSDL Document Example

Following is the WSDL file that is provided to demonstrate a simple WSDL program.

Assuming the service provides a single publicly available function, called *sayHello*. This function expects a single string parameter and returns a single string greeting. For example if you pass the parameter *world* then service function *sayHello* returns the greeting, "Hello, world!".

Content of HelloService.wsdl file

First set of lines of WSDL doc

```
<definitions name="HelloService" targetNamespace="uri"
              xmlns="uri" xmlns:soap="uri"
              xmlns:tns="uri"
              xmlns:xsd="uri">
```

Second set of lines of WSDL doc

```
<types>
  <schema>
    <import namespace="uri" location="uri"/>
    <complexType name=" ">
      <sequence>
        <element name=" " type=" " />
        :
        :
      </sequence>
    </complexType>
  </schema>
</types>
```

Third set of lines of WSDL doc

```
<message name="SayHelloRequest">
  <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
  <part name="greeting" type="xsd:string"/>
</message>
```

Fourth set of lines of WSDL doc

```

<portType name="Hello_PortType">
  <operation name="sayHello">
    <input message="tns:SayHelloRequest"/>
    <output message="tns:SayHelloResponse"/>
  </operation>
</portType>

```

Fifth set of lines of WSDL doc

```

<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding style="rpc" transport="uri"/>
  <operation name="sayHello">
    <soap:operation soapAction="sayHello"/>
    <input
      use="encoded"/>
    </input>
    <output
      <soap:body encodingStyle="uri" namespace="uri"
      use="encoded"/>
    </output>
  </operation>
</binding>

```

The last set of lines of WSDL doc

```

<service name="Hello_Service">
  <port binding="tns:Hello_Binding" name="Hello_Port">
    <soap:address location="uri">
  </port>
</service>
</definitions>

```

Analysis of the Example

- Definition : HelloService
 - Type : Using built-in data types and they are defined in XMLSchema.
 - Message :
 1. sayHelloRequest : firstName parameter
-

- 2. sayHelloresponse: greeting return value
- Port Type: sayHello operation that consists of a request and response service.
- Binding: Direction to use the SOAP HTTP transport protocol.
- Service: Service available at the URI
- Port: Associates the binding with the URI where the running service can be accessed.

A detailed description of these elements is given in subsequent sections

WSDL Definition Element

The <definition> element must be the root element of all WSDL documents. It defines the name of the web service.

Here is the example piece of code from last session which uses *definition* element.

```
<definitions name="HelloService"      targetNamespace="uri"
                                xmlns="uri"
                                xmlns:soap="uri"
                                xmlns:tns="uri"
                                xmlns:xsd="uri">
    .....
</definitions>
```

The definitions element is a container of all the other elements.

The definitions element specifies that this document is the *HelloService*.

The definitions element specifies a *targetNamespace* attribute. The *targetNamespace* is a convention of XML Schema that enables the WSDL document to refer to itself.

It also specifies numerous namespaces that will be used throughout the remainder of the document.

WSDL Types Element

A Web service needs to define its inputs and outputs and how they are mapped into and out of services. WSDL <types> element takes care of defining the data

types that are used by the web service. In other words, the types element describes all the data types used between the client and server. The types element in WSDL doc is optional.

WSDL Message Element

- The <message> element describes the data being exchanged between the Web service providers and consumers.
- Each Web Service has two messages: input and output.
- The input describes the parameters for the Web Service and the output describes the return data from the Web Service.
- Each message contains zero or more <part> parameters, one for each parameter of the Web Service's function.
- Each <part> parameter associates with a concrete type defined in the <types> container element.

Let's take a piece of code from the Example:

```
<message name="SayHelloRequest">
  <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse"> <part
name="greeting" type="xsd:string"/>
</message>
```

Here, two message elements are defined. The first represents a request message *SayHelloRequest*, and the second represents a response message *SayHelloResponse*.

Each of these messages contains a single part element. For the request, the part specifies the function parameters; in this case, we specify a single firstName parameter. For the response, the part specifies the function return values; in this case, we specify a single greeting return value.

WSDL portType Element

<portType> can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations.

Let's take a piece of code from the Example:


```

<portType name="Hello_PortType">
  <operation name="sayHello">
    <input message="tns:SayHelloRequest"/>
    <output message="tns:SayHelloResponse"/>
  </operation>
</portType>

```

- The portType element defines a single operation, called *sayHello*.
- The operation itself consists of a single input message *SayHelloRequest*
- The operation itself consists of a single output message *SayHelloResponse*
- *tns* prefix indicates a namespace called *targetNameSpace*

WSDL Binding Element

- The <binding> element provides specific details on how a *portType* operation will actually be transmitted over the wire.
- The bindings provide concrete information on what protocol is being used to transfer *portType* operations.
- The bindings provide information where the service is located.
- For SOAP protocol, the binding is <soap:binding>, and the transport is SOAP messages on top of HTTP protocol.

soap:binding

This element indicates that the binding will be made available via SOAP. The *style* attribute indicates the overall style of the SOAP message format. A style value of *rpc* specifies an RPC format. The *transport* attribute indicates the transport of the SOAP messages.

soap:operation

This element indicates the binding of a specific operation to a specific SOAP implementation. The *soapAction* attribute specifies that the SOAPAction HTTP header be used for identifying the service.

soap:body

This element enables you to specify the details of the input and output messages. In the case of HelloWorld, the body element specifies the SOAP encoding style and the namespace URN associated with the specified service.

Here is the piece of code from Example:

```

<binding name="Hello_Binding" type="tns:Hello_PortType">
  <soap:binding style="rpc"

```

```

        transport="uri"/>
<operation name="sayHello">
  <soap:operation soapAction="sayHello"/>
  <input>
    <soap:body encodingStyle="uri" namespace="uri" use="encoded"/>

  </input>
  <output>
    <soap:body encodingStyle="uri" namespace="uri" use="encoded"/>

  </output>
</operation>
</binding>

```

WSDL Service Element

The <service> element defines the ports supported by the Web service. For each of the supported protocols, there is one port element. The service element is a collection of ports.

Web service clients can learn from the service element where to access the service, through which port to access the Web service, and how the communication messages are defined.

A <port> element defines an individual endpoint by specifying a single address for a binding.

Here is the pice of code from Example:

```

<service name="Hello_Service">
  <documentation>WSDL File for HelloService</documentation>
  <port binding="tns:Hello_Binding" name="Hello_Port">
    <soap:address
      location="http://www.examples.com/SayHello/">
    </port>
  </service>

```

XML Schema

The purpose of an XML Schema is to define the legal building blocks of an XML document. It is used to represent data types of an XML document. It is an alternative to DTD (Document Type Definition). XML Schema defines elements, attributes and values of elements and attributes.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas Data Types

One of the greatest strength of XML Schemas is the support for data types. Using XML schema it is easier to describe allowable document content; it is easier to validate the correctness of data; it is easier to convert data between different data types.

Built in data types supported by XML schema

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

XML schema defines in the namespace *http://www.w3.org/2001/XMLSchema* that contains built in data types. There are two classes of XML schema data types:

- Complex type is a data type is represented using markup.
- Simple type is a data type whose values are represented in XML doc by character data.

XML markup such as <types> is known as XML schema that conforms to W3C defined XML schema vocabulary which defines all or part of the vocabulary for another XML document. The <schema> is the root element for any XML schema document. The child elements of schema define the data types.

Example for writing a simple XML Schema

Step 1: Write a xsd file in which the desired structure of the XML document is defined and named it as StudentSchema.xsd.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="student">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" value="xs:string"
            />
                <xs:element name="regno" value="xs:string" />
                <xs:element name="dept" value="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The xs qualifier used to identify the schema elements and its types. The xs:schema is the root element. It takes the attributes xmlns:xs which has the value, "http://www.w3.org/2001/XMLSchema. This declaration indicates that the document follows the rule of XMLSchema defined by W3 in year 2001.

The xs:element is used to define the xml element. The Student element is complexType which has three child elements: name, regno and dept. All these elements are of simple type string.

Step 2: Write a XML document and reference the xsd file. Named it as myXML.xml

```
<?xml version="1.0"?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="StudentSchema.xsd">
    <name> Raj </name>
    <regno>3212654556</regno>
    <dept>CSE</dept>
```

</student>

The following fragment: `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` tells the XML Schema Instance namespace available.

The following code, `xsi:schemaLocation="StudentSchema.xsd"` is used to reference the XML schema document (xsd) file which contains the definition of thisxml document.

Various xml validation tools are available which can be used to validate xml and its schema document.

Advantages

- Supports data types
- Supports namespaces
- XML schemas support a set of data types, similar to the ones used in most common programming languages
- Provides the ability to define custom data types
- Easy to describe allowable content in the document
- Easy to validate the correctness of data
- Object oriented approach like inheritance and encapsulation can be used in creating the document
- Easier to convert data between different data types
- Easy to define data formats
- Easy to define restrictions on data
- It is written in xml so any xml editor can be used to edit xml schema
Xml Parser can be used to parse the schema file
- XML schemas are more powerful than DTDs. Everything that can be defined by the DTD can also be defined by schemas, but not vice versa.

Disadvantages

- Complex to design and learn
- Maintaining XML schema for large XML document slows down the processing of XML document

DTD for the above xml file instead of XML schema

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.

A DTD defines the document structure with a list of legal elements and attributes.

Step 1: Create a DTD. Name it as student.dtd

```
<!ELEMENT student(name, regno, dept)>
<!ELEMENT  name(#PCDATA)>
<!ELEMENT  regno(#PCDATA)>
<!ELEMENT dept(#PCDATA)>
!ELEMENT student defines that the note element contains three elements: "name, regno, dept"
```

PCDATA means parsed character data which is the text found between the start tag and the end tag of an XML element.

Step 2: Write the XML document and reference the DTD file in it. Name it as myXML.xml

```
<?xml version="1.0"?>
<!DOCTYPE student SYSTEM student.dtd>
<student>
    <name> Raj </name>
    <regno>3212654556</regno>
    <dept>CSE</dept>
</student>
```

!DOCTYPE student defines that the root element of this document is student and links the myXML.xml file with the student.dtd file.

SOAP

SOAP is an acronym for Simple Object Access Protocol. SOAP defines a protocol for message exchange between applications. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

- SOAP describes envelope and message formats, and has a basic request/response
- SOAP is a communication protocol

- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP communicates via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP is a W3C recommendation

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information

Structure of SOAP Message

HTTP binding

// used to direct the message

Envelope

// helps the server to identify the message as SOAP

Header

// Optional. It adds features to SOAP message such as authentication, message route etc.

Body

// contain actual message

The following code fragment gives an outline of the SOAP message

```
// code for HTTP binding
```

```
<soap:Envelope xmlns:soap="uri" soap:encodingStyle="uri">
  <soap:Header>
    ... ..
  </soap:Header>

  <soap:Body>
    ... ..
    <soap:Fault>
      ... ..
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

- The SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message. The xmlns:soap namespace defines the Envelope as a SOAP Envelope. The encodingStyle attribute is used to define the data types used in the document.
- The SOAP Header element contains header information. If the Header element is present, it must be the first child element of the Envelope element.
- The SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.
- The SOAP Fault element holds errors and status information for a SOAP message.

A SOAP Example

In the example below, a GetStockPrice request is sent to a server. The request has a StockName and Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".

A SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: n bytes
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="uri" soap:encodingStyle="uri">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
```



```
        <m:StockName>IBM</m:StockName>
      </m:GetStockPrice>
    </soap:Body>
  </soap:Envelope>
```

The SOAP response:

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: n bytes

```
<?xml version="1.0"?><soap:Envelope xmlns:soap="uri" soap:encodingStyle="uri">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP and HTTP

- A SOAP method is an HTTP request/response that complies with the SOAP encoding rules. SOAP message is HTTP + XML = SOAP.
- A SOAP request could be an HTTP POST or an HTTP GET request.

Java support for SOAP

Java provides support for creating and manipulating SOAP documents through SAAJ which is an acronym of Soap with Attachments Api for Java. This API is part of JWS DP.

Related technologies

- The alternative for creating web services using JWS DP is sun J2EE platform, Apache Axis.
- Also, IBM, WebSphere, BEA and WebLogic development platforms are alternative tools for developing java based web services.
- Personal HomePage (PHP) development platform can also be used for creating web services. It contains PEAR (Personal home page Extension and Application Repository) for parsing a WSDL document.

- UDDI (Universal Discovery Description and Integration) is used to register and locate web services. Alternative to UDDI is JAXR (Java Api for Xml Registry)
- WS-I Basic Profile (Web Service Interoperability Basic Profile) is a tool which helps the developers how to use various web service technologies.

UNIT – I

1. What is the difference between node and host?

A node is any addressable device connected to a network whereas the host is a more specific descriptor that refers to a networked general-purpose computer rather than a single purpose device (such as a printer).

2. What is the purpose of routers?

Router operates like electronic postal workers that evaluate and forward packets between networks.

3. Define protocol.

A protocol is a formal set of rules that must be followed in order to communicate.

4. Why are the protocols layered?

Layering protocols simplifies the task of communicating over the network and it allows for reuse of layers that are not specific to a particular application.

5. Define encapsulation.

Placing the data inside a package of headers is known as encapsulation.

6. Define port.

A port is a logical channel to an application running on a host. ie., The applications running on the host machines are uniquely identified by port numbers.

7. What do you mean by well-known ports?

Port numbers can range from 1 to 65535, however ports 1 to 1023 are reserved. These reserved ports are referred to as well-known ports because the Internet Assigned Numbers Authority publicly documents the applications that use them.

8. What is meant by Name Resolution?

Name Resolution is the process of mapping a hostname to its corresponding IP Address. One way to translate a hostname to an IP address is to look it up in a simple text file. The second way is the domain name service, which is a distributed database containing all registered hostnames on the Internet and their IP addresses.

9. Define protocol tunneling.

Protocol tunneling is the process of encapsulating one protocol within another protocol that operates on the same layer.

10. Define URI, URL, URN.

1. URI (Uniform Resource Identifier): It identifies an object on the Internet.
2. URL (Uniform Resource Locator): It is a specification for identifying an object

such as a file, newsgroup, CGI program or e-mail address by indicating the exact location on the internet.

3. URN (Uniform Resource Name): It is a method for referencing an object without declaring the full path to the object.

11. What are the components of HTTP URL?

The components are host, an optional port, path, filename, section and query string.

12. Define URL encoding.

URL encoding involves replacing all unsafe and nonprintable characters with a percent sign (%) followed by two hexadecimal digits corresponding to the character's ASCII value.

13. What are the issues of next generation IP?

The issues to be considered in IP next generation

are I. Addresses Space Growth

2. Support large Global networks

3. A clear way of transition from the existing IP to new IP next generation

14. What is the difference between TCP and UDP?

TCP:

Connection oriented transport protocol

Sends data as a stream of bytes

Guarantee of delivery

UDP:

Connection less protocol

Datagram service

No guarantee of delivery.

15. What does ICMP provide?

ICMP provides

1. Error messaging

2. Demand reply functions

16. Define IGMP.

It is Internet Group Management protocol. It provides

1. Broadcasting

2. Multicasting

17. List the goals of SGML.

1. To manage the flow of millions of pages.

2. For structuring information exchange

3. For modeling inter-document linkages

4. For managing information flows between departments and weapons systems

18. What is the role of server?

1. Manages application tasks
2. Handles storage
3. Handles security
4. Provides scalability
5. Handles accounting and distribution

19. What are the necessities of using HTML forms?

1. Gathering user information
2. Conducting Surveys
3. Interactive services

20. What are the sequences of steps for each HTTP request from a client to the server?

1. Making the connection
2. Making a request
3. The response
4. Closing the connection

21. Define MIME.

MIME (Multipurpose Internet Mail Extensions) is an open standard for sending multipart, multimedia data through Internet email.

22. List the predefined MIME content types.

1. Text
2. Multipart
3. Message
4. Image
5. Audio
6. Video
7. Model
8. Application

23. Define HTML.

It is a simple page description language, which enables document creation for the web.

24. What is meant by loop back address?

A zone that enables the server to direct traffic to itself. The host number is almost always 127.0.0.1.

UNIT II

25. Define CGI -Common Gateway Interface.

A specification for transferring information between a World Wide Web server and a CGI program. A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic.

26. Write a note on Internet Information Server (IIS).

Microsoft's Web server that runs on Windows NT platforms, In fact, IIS comes bundled with Windows NT 4.0. Because IIS is tightly integrated with the operating system, it is relatively easy to administer. However, currently IIS is available only for the Windows NT

platform, whereas Netscape's Web servers run on all major platforms, including Windows NT platform, OS/2 and UNIX.

27. What are ISAPI (Internet Server API) and NSAPI (Netscape Server API)

ISAPI (Internet Server API): An API for Microsoft's IIS (Internet Information Server) Web server. ISAPI enables programmers to develop Web-based applications that run much faster than conventional CGI programs because they're more tightly integrated with the Web server. In addition to IIS, several Web servers from companies other than Microsoft support ISAPI.

NSAPI -Netscape Server API: An API for Netscape's Web servers. NSAPI enables programmers to create Web-based applications that are more sophisticated and run much faster than applications based on CGI scripts.

28. What is API -Application Program Interface?

A set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a program by providing all the building blocks. A programmer puts the blocks together.

Most operating environments, such as MS- Windows, provide an API so that programmers can write applications consistent with the operating environment. Although APIs are designed for programmers, they are ultimately good for users because they guarantee that all programs using a common API will have similar interfaces. This makes it easier for users to learn new programs.

29. What are Servlets?

A small program that runs on a server, the term usually refers to a Java applet that runs within a Web server environment. This is analogous to a Java applet that runs within a Web browser environment.

Java servlets are becoming increasingly popular as an alternative to CGI programs. The biggest difference between the two is that a Java applet is *persistent*. This means that once it is started, it stays in memory and can fulfill multiple requests. In contrast, a CGI program disappears once it has fulfilled a request. The persistence of Java applets makes them faster because there's no wasted time in setting up and tearing down the process.

30. What are Applets?

A program designed to be executed from within another application. Unlike an application, applets cannot be executed directly from the operating system. With the growing popularity of OLE (object linking and embedding), applets are becoming more prevalent. A well-designed applet can be invoked from many different applications.

Web browsers, who are often equipped with Java virtual machines, can interpret applets from Web servers. Because applets are small in files size, cross-platform compatible, and highly secure (can't be used to access users' hard drives), they are ideal for small Internet applications accessible from a browser.

31. What do you mean by Server-side?

Occurring on the server side of a client-server system. For example, on the World Wide Web, CGI scripts are server-side applications because they run on the Web server. In contrast, JavaScript scripts are client-side because they are executed by your browser (the client). Java applets can be either server-side or client-side depending on which computer (the server or the client) executes them.

32. What is a protocol?

An agreed-upon format for transmitting data between two devices. The protocol determines the following:

1. The type of error checking to be used
2. Data Compression method, if any
3. How the sending device will indicate that it has finished sending a message
4. How the receiving device will indicate that it has received a message

33. What is ActiveX?

A loosely defined set of technologies developed by Microsoft for sharing information among different applications. ActiveX is an outgrowth of two other Microsoft technologies called *OLE (Object Linking and Embedding)* and *COM (Component Object Model)*. As a moniker, *ActiveX* can be very confusing because it applies to a whole set of COM-based technologies. Most people, however, think only of ActiveX controls, which represent a specific way of implementing ActiveX technologies.

34. Write a note on ActiveX controls.

A control using ActiveX technologies. An ActiveX control can be automatically downloaded and executed by a Web browser. ActiveX is not a programming language, but rather a set of rules for how applications should share information. Programmers can develop ActiveX controls in a variety of languages, including C, C++, Visual Basic, and Java. An ActiveX control is similar to a Java applet. Unlike Java applets, however, ActiveX controls have full access to the Windows operating system. This gives them much more power than Java applets, but with this power comes a certain risk that the applet may damage software or data on your machine. To control this risk, Microsoft developed a registration system so that browsers can identify and authenticate an ActiveX control before downloading it. Another difference between Java applets and ActiveX controls is that Java applets can be written to run on all platforms, whereas ActiveX controls are currently limited to Windows environments. Related to ActiveX is the scripting language *VBScript* that enables Web authors to embed interactive elements in HTML documents.

35. Explain about HTTP Connection.

It is a communication channel between web browser and web server. It begins on the client side with the browser sending a request to the web server for a document.

Request Header Fields are

1. From
2. Reference
3. If_modified_since
4. Pragma
5. User Agent

36. What is meant by Stateless Connection?

When a web server receives a HTTP request from a web browser it evaluates the request and returns the requested document, if it exists, and then breaks the HTTP connection. This document is preceded by the response header, which has details about how to display the document that will be sent by the server. Each time a request is made to the server, it is as if

there was no prior connection and each request can yield only a single document. This is known as Stateless Connection.

37. Write a note on Environment variables.

In CGI, the server prepares the environment variables before it launches the CGI script. These represent the current state of the server that is asking for the information. The environment variables are not set from the command line but are created on the fly, and lasts only until that particular script is finished. Each script gets its own unique set of variables and multiple scripts can be executed at once, each in its own environment.

38. What are STDIN and STDOUT?

These are mnemonics for standard input and standard output, two predefined stream / file handles. Each process already inherits these two handles already open. From the script's point of view, STDIN is what comes from the browser via the server when the post method is used, and the STDOUT is where it writes its output back to the browser. The script picks up the environment variables and reads STDIN as appropriate. It then does whatever it was designed to do and writes its output to STDOUT.

39. What are the two commonly used Request methods?

The request methods tell the script how it was invoked. Based on this information, the script decides how to act. The request method is passed to the script using environment variable called REQUEST- METHOD. The two most common request methods used are GET and POST.

GET

GET is a request for data, the same method used for obtaining static documents. The GET method sends request information as parameter tacked onto the end of the URL. These parameters are passed to the CGI program in the environment variable QUERY-STRING.

E.g.: If the script is called myprog.exe and is invoked from a link with the form `<A HREF="cgi-bin/myprog.exe? lname=blow&fname=joe">` The REQUEST_METHOD will be the string GET, and the QUERY_STRING will contain lname=blow & fname=joe. A mandatory question mark separates the name of the script from the beginning of the QUERY_STRING. If a slash is used instead of the question mark; the server passes the information to script using the PATH_INFO variable instead of the QUERY_STRING variable.

POST

POST operation occurs when the browser sends data from a fill -in form to the server. With POST, the QUERY_STRING mayor may not be blank, depending on the server. The data from a POSTed query gets passed from the server to the script using STDIN. Because STDIN is a stream and the script needs to know how much valid data is waiting, the server also Supplies another variable, CONTENT_LENGTH, to indicate the size in bytes of the incoming "data".

The format for POSTed data is
Variable1=value1 & variable2=value2 &etc.

After the required data is available, the script executes and writes its output to the STDOUT. The MIME code that the server sends to the browser indicates the type of the file that is being sent. This information that precedes the file is usually called the *header*. Since the script generates the output on the fly the server will not be able to create a header for that information. Hence this information has to be supplied by the script itself. Failure will result in the browser receiving information that it does not know how to display.

40. Explain about URL Encoding.

HTTP specification requires that the URL data should be encoded in such a way that it can be used on almost any hardware and software platforms. Information specified in this way is called *URL encoded*. If parameters are passed as a part of query string or path information, they will take the form of 'Name-Value' pairs.

variable=value1&variable2=value2& so on for each variable defined in the form.

The variables or name value pairs are separated by ©&©.Real ampersand is escaped -that is, encoded as a two-digit hexadecimal value representing the character. Escaped characters are indicated in URL-encoded string by the percent (%) sign. Blank spaces are replaced by ©+©sign. Before the script can deal with the data it has to parse and decode it. The script scans through the string looking for an ampersand. When it is found the string is broken from that point. The variable©s name is every thing up to the equal sign in the string and the value is every thing after the equal sign. The script continues to parse the original string for the next ampersand, and so on until the original string is exhausted. After the variables are separated, they are decoded as follows.

1. Replace all plus signs with blank spaces.

2. Replace all %## (Percent sign followed by two hexadecimal digits) with the corresponding ASCII character.

Separate the name-value pairs from the URL and store the values separately.

41. List the advantages of CGI scripting?

1. CGI programs are relatively safe to run.
2. A CGI program can crash without damaging the server, since it only has limited access to the server.
3. Reduces the burden of server
 - a. Sends prepared messages / mails e customer reply
 - b. Capability to process forms and prepares output based on form input.
 - c . Hit counts / Page counters.

42. Explain about Session tracking.

A session is basically a conversation between a browser and a server. All the above technologies can save information for the current session for a particular user visiting a site. The session is important, as H1TP is a stateless protocol. This means that the connection between web server and a web browser is not automatically maintained, and that the state of a web session is not saved.

State is a general term that includes "everything about your situation" and the specifics vary based on the application. In a word processor, the state of the application would include which windows are open, where they are on the screen, and what files you most recently used. In a web application, the state would include any data that you had entered, the results of any queries that you had run, and your security access information (e.g. whether you have logged in to the site).

UNIT III

43. Define packet switched networks.

Packet switched network means that data traveling on the network is broken into chunks called packets and each packet is handled separately.

44. Define socket.

The socket is a software abstraction used to represent the terminals of a connection between two machines or processes.

45. What are the basic operations of client sockets? Connect to a remote machine
Send data Receive data Close a connection

46. What are the basic operations of Server socket? Bind to a port
Listen for incoming data
Accept connections from remote machines on the bound port

47. List all the socket classes in java.
Socket
ServerSocket
Datagram Socket
Multicast Socket
Secure sockets

48. What the Socket Object does?

Socket object is the java representation of a TCP connection when a socket is created; a connection is opened to the specified destination.

49. What is meant by Server Socket?

ServerSocket represents a listening TCP connection. Once an incoming connection is requested, the ServerSocket object will return a Socket object representing the connection.

50. What do you mean by DatagramSocket and DatagramPacket?

DatagramSocket represents a connectionless datagram socket. This class works with the DatagramPacket class to provide for communication using the UDP protocol.

51. Write a note on Connect Exception.

This exception is raised when a connection is refused at the remote host. (ie, no process is listening on that port).

52. What is a multicast socket?

Multicasting sends data from one host to many different hosts, which are in the multicast group.

53. What is multicast address and the range of address?

A multicast address is the address of a group of hosts called a multicast group. Multicast addresses are IP addresses in the range 224.0.0.0 to 239.255.255.255

54. What are the different types of IP addresses?

Unicast address: It is used for transmitting a message to single destination node
Multicast address: It delivers a message to a group of destination nodes, which are necessarily in the same sub network.

Broadcast address: It transmits a message to all nodes in a sub network.

55. What is meant by protocol handler?

Protocol handlers are used to retrieve the web objects using application specific protocols. The protocols are specified in the URL referencing the object.

56. How are the protocol handlers implemented?

Four different classes in the java.net package implement the protocol handlers:

1. URL
2. URLStreamHandler
3. URLConnection
4. URLStreamHandlerFactory

57. What are the methods for parsing URLs?

1. parseURL(URL u, String spec, int start, int limit)- splits the URL into parts
2. setURL(URL u, String protocol, String host, int port, String file, String ref) - assigns values to the URL's fields.

58. What is content handler?

Content handlers are used to retrieve objects via an URLConnection object.

59. What is Remote Method Invocation?

The Remote Method Invocation is application-programming interface that allows java objects on different hosts communicate with each other.

60. What do you mean by remote object?

Objects that have methods that can be called across virtual machines are remote objects.

61. Define serialization.

It is the process of converting a set of object instances that contain references to each other into a linear stream of bytes, which can then be through a socket. It is the mechanism used by RMI to pass objects between Java Virtual Machines.

62. What are the responsibilities of stub?

A stub for a remote object is the client side proxy for the remote object. A client side stub is responsible for:

1. Initiating a call to the remote object
2. Marshaling arguments to a marshal stream

3. Informing the remote reference layer that the call should be invoked
4. Unmarshaling the return value or exception from a marshal stream

63. What is the role of skeleton in RMI?

A skeleton for a remote object is a server side entity that contains a method which dispatches calls to the actual remote object implementation. The skeleton is responsible for

1. Unmarshaling arguments from the marshal stream.
2. Making the up-call to the actual remote object.
3. Marshalling the return value of the call to an exception onto the Marshall stream

64. List down the layers of RMI architecture.

1. Stubs/Skeletons
2. Remote reference layer
3. Transport layer

65. Define Object Activation.

Object Activation is mechanism, which allows a java object to be bound and then activated at some later data simply by referencing the object through the Registry.

66. Write down the Socket object methods to get information about a socket.

1. `getInetAddress ()` - displays which remote host the Socket is connected to
2. `getPort ()` - displays which port the Socket is connected to on the remote host.
3. `getLocalPort ()` - to find the port number for the local end of a connection
4. `getLocalAddress ()` - tells you which network interface a socket is bound to.

67. What operations Multicast Socket Perform?

1. Join a multicast group
2. Send data to the members of the group
3. Receive data from the group
4. Leave the multicast group

UNIT IV

68. What are Style Sheets?

Style sheets are collections of style information that are applied to plain text. Style information includes font attributes such as type size, special effects (bold, italic, underline), color and alignment. Style sheets also provide broader formatting instructions by specifying values for quantities such as line spacing and left and right margins.

69. List down the ways of including style information in a document.

1. Linked Styles - Style information is read from a separate file that is specified in the `<LINK>` tag
2. Embedded Styles - Style information is defined in the document head using the `<STYLE>` and `</STYLE>` tags.
3. Inline Styles - Style information is placed inside an HTML tag and applies to all content between that tag and its companion closing tag.

70. Define cascading.

Cascading refers to a certain set of rules that browsers use, in cascading order, to determine how to use the style information. Such a set of rules is useful in the event of conflicting style information because the rules would give the browser a way to determine which style is given precedence.

71. What are the style precedence rules when using multiple approaches?

Inline styles override both linked style sheets and style information stored in the document head with <STYLE> tag.

Styles defined in the document head override linked style sheets.

Linked style sheets override browser defaults.

72. Give the syntax to specify a characteristic in linked style sheet.

{Characteristic: value}

Multiple characteristic/value pairs should be separated by semicolons.

73. List down font characteristics permitted in style sheets.

- 1.font-family
- 2.font-size
- 3.font-weight
- 4.font-style
- 5.font-variant

74. Write a note on content positioning characteristic "Visibility".

Enables the document author to selectively display or conceal positioned content; Possible values are show or hide.

75. Define XML.

XML is a meta-markup language that provides a format for describing structured data. This facilitates more structured declarations of content and more meaningful search results across multiple platforms.

76. Define DTD.

A DTD is a set of rules that specifies how to use XML markup. It contains specifications for each element, including what the element's attributes are, what values the attributes can take on and what elements can be contained in others.

77. What are the XML rules for distinguishing between the content of a document and the XML markup element?

- 1.The start of XML markup elements is identified by either the less than symbol (<) or the ampersand (&) character
- 2.Three other characters, the greater than symbol (>), the apostrophe or single quote (') and the double quotation marks (") are used by XML for markup.
- 3.To use these special characters as content within your document, you must use the corresponding general XML entity.

78. Define scriptlets.

Scriptlets enable you to create small, reusable web applications that can be used in any web page. Scriptlets are created using HTML, scripting and Dynamic HTML. To

include them in an HTML document use the <OBJECT> tag.

79. Define ASP.

Active Server Pages (ASP) is a server-side scripting technology that can be used to create dynamic and interactive web applications.

80. What are the ASP objects?

- 1.Application -It manages your web application.
- 2.Session -It manages and tracks individual user sessions.
- 3.Server -It controls behavior of your web server
- 4.Response -It transmits information from the web server to web browser
- 5.Request -It retrieves information from the browser for processing at the server.

81. What is global.asa file?

The global.asa file is a Active Server Application file you can track and manage the application and session events, variables and objects. When you start the application the server will load the global.asa file into memory.

82. Define response object and list its methods.

The response object transmits information from the web server to browser. Methods are:

1. Write
- 2.BinaryWrite
- 3.Redirect
- 4.AppendToLog
- 5.AddHeader
- 6.Clear
- 7.Flush

83. Define JSP.

Java Server Pages (JSP) are simple technology used to generate dynamic HTML on the server side.

84. Define Directives.

Directives are JSP elements that provide global information about an entire JSP page.

85. Write down the various attributes for the page directives in JSP.

The page directive defines information that will be globally available for that Java Server Page,

1. language
2. extends
3. import
4. session
5. buffer
6. contenttype

86. What is meant by firewall?

A firewall is a piece of network hardware that serves as a secure gateway between

an internal network and the Internet. It protects the internal network from unauthorized access or activity,

87. Write a note on proxy server.

A proxy server is a host that makes Internet request on behalf of other machines on the network, Proxy servers are often used to cache frequently requested files or to monitor Internet use within a Corporation.

88. What does DHTML refer?

DHTML refers to collection of technologies, which makes HTML documents more dynamic and interactive.

89. Define SSI.

Server Side Includes (SSI) gives you a way to insert the content of another file into a file before the web server processes it.

90. What does data binding mean?

Data binding is DHTML feature that lets you easily bind individual elements in your document to data from another source such as database or comma delimited text file.

UNIT- 5

ON LINE APPLICATION

91. What is meant by Plug-in?

A hardware or software module that adds a specific feature or service to a larger system. The idea is that the new component simply *plugs in* to the existing system. For example, there are number of plug-ins for the Netscape Navigator browser that enable it to display different types of audio or video messages. Navigator plug-ins are based on MIME file types.

92. What do you mean by JDBC?

JDBC Part of the Java Development Kit which defines an application-programming interface for Java for standard SQL access to databases from Java programs.

93. Define ODBC.

It is a standard for accessing different database systems. There are interfaces for Visual Basic, Visual C++, SQL and the ODBC driver pack contains drivers for the Access, Paradox, dBase, Text, Excel and Btrieve databases.

94. List any two keyboard events?

1. onKeyPress
2. onKeyUp
3. onKeyDown

95. List any two mouse events?

1. onMouseUp
2. onMouseDown
3. onMouseOver
4. onClick

96. Define virtual organization.

The virtual organization is defined as being closely coupled upstream with its suppliers and downstream with its customers such that where one begins and the other ends means little to those who manage the business processes within the entire organization.

97. List the major approaches to form virtual organization?

1. Downward networking: a large, vertically integrated company seeking to reduce its overhead by outsourcing initiates it

2. Lateral Approach: It is observed in small, specialized firms that in the interest of seeking strategic alliances, form partnerships along a value-added chain.

98. What do mean by search engine?

It is a program or web page that enables you to search an Internet site for a specific keywords or words.

99. List the features of online shopping.

1. Make it easy for you to browse and purchase as much as possible from their websites.

2. Products displayed from the online store are associated with links to detailed descriptions of the products

3. You are able to compare the product to similar products based on the features and pricing.

4. It also brings you to the web sites of the product manufacturer for more information.

5. Shopping sites want you to have a personal account created before you shop their site. The account typically provides the business with your name, address, e-mail, phone number and possibly credit card numbers.

6. Some online shopping sites offer free samples. For example, online music stores sometime have audio samplings of the CDs they sell

100. How do search engine work?

When you enter a keyword, the search engine examines its online database and presents to you a listing of sites that, in theory, match your search criteria.

PART B

1. Explain in detail the working of the TCP/IP and HTTP protocols
2. Explain in detail about web client and web-server communication
3. Briefly explain any five XHTML controls
4. Explain about HTML elements in detail
5. How do you use frame collections to access objects in a separate frame on your webpage
6. Explain how functions can be written in Java script with an example
7. Explain any eight CSS text properties in detail
8. Explain the way in which java script handles arrays with example

9. State and explain the types of statements in java script?
10. Explain about the document tree in detail
11. Describe the servlet architecture and the various information invoked by the servlet container.
12. Explain DOM event handling in detail
13. What is a session? Explain how client state is maintained using session and also explain about session tracking and session management using an example

14. List the XML syntax rules in detail and Explain how a XML document can be displayed on a browser
15. Consider the database table with the following structure Student (studentname, register_number, Grade_obtained, Age) Write a JSP to display all the details of the Student
16. Explain the model view controller architecture pattern in detail
17. What does JSP scripting component includes? Explain with program
18. Explain how XML is processed with the help of SAX
19. Write a DTD for the following schema (emp_id, emp_name (firstname, lastname), dob(dd, mm, yyyy). address(city, state)).
20. Explain the SOAP elements and JAX RPC in detail
21. Explain in detail the XML schema, built in and user defined data type in detail
22. Explain the JDBC database connectivity in detail
23. Explain in detail the XML schema,built in and user defined data types in detail

