

SATHYABAMA UNIVERSITY

(Established under Section 3, UGC Act 1956)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



SCSX1016 Web Technology

SCSX1016 Web Technology

Unit-1

Introduction

Web technology is the establishment and use of mechanisms that make it possible for different computers to communicate and share resources.

According to Microsoft.com, some examples of web technologies include:

- 1) Mark-up languages including HTML, XML.
- 2) Programming Languages and Technologies which help in creating applications for the web: some of the languages are Perl, C#, Java and Visual Basic .Net.
- 3) Web servers and server technologies which facilitate request handling on a network where different users have to share the same resources and communicate with one another.
- 4) Databases, which are crucial for data and information storage on a computer network.
- 5) Business applications customized for specific execution of tasks on a network.

Network Concepts

A network is a communication system that allows users to access resources on other computers and exchange messages with one another. It allows users to share resources on their own system or access shared resources on other systems.

Web technologies are infrastructural building blocks of any effective computer network: Local Area Network (LAN), Metropolitan Area Network (MAN) or a Wide Area Network (WAN), such as the Internet. Communication on a computer could never be as effective as they are without the plethora of web technologies in existence.

Web technologies played an important part in redefining access to data, the sharing of data, and the design of network applications. An *intranet* is an internal network built with Web technologies while an *extranet* is a business-to-business network with links across the Internet.

Standard protocols:

- TCP
handles conversion between messages and streams packets
- IP
handles addressing of packets across networks
- TCP/IP

enables packets to be sent across multiple networks using multiple standards

- Telnet

One of the earliest standards for exchanging transmission, directly connect accounts on different systems.

- SMTP

Specifies another way of direct connection

- MIME

Extension to the SMTP Protocol which supports the exchange of richer data files such as audio-, video-, and images data.

- FTP

Supports file transfer between Internet sites and allows a system to publish a set of files by hosting an FTP sever.

Web concepts:

What is the web?

The web is a whole bunch of interconnected computers talking to one another. The computers (on the web) are typically connected by phone lines, digital satellite signals, cables, and other types of data-transfer mechanisms. A 'data-transfer mechanism' is a way to move information from point A to point B to point C and so on.

The computers that make up the web can be connected all the time, or they can be connected only periodically. The computers that are connected all the time are typically called a 'server'. Servers are computers just like the one you're using now to read this notes, with one major difference, they have a special software installed called 'server' software.

What is Internet?

The Internet is essentially a global network of computing resources. We can think of the Internet as a physical collection of routers and circuits as a set of shared resources.

Some common definitions given in the past include:

- A network of networks based on the TCP/IP communications protocol.
- A community of people who use and develop those networks.

Internet-Based Services

Some of the basic services available to Internet users are –

- **Email** – A fast, easy, and inexpensive way to communicate with other Internet users around the world.
- **Telnet** – Allows a user to log into a remote computer as though it were a local system.
- **FTP** – Allows a user to transfer virtually every kind of file that can be stored on a computer from one Internet-connected computer to another.

- **UseNet news** – A distributed bulletin board that offers a combination news and discussion service on thousands of topics.
- **World Wide Web (WWW)** – A hypertext interface to Internet information resources.

What is WWW?

WWW stands for **World Wide Web**. A technical definition of the World Wide Web is – All the resources and users on the Internet that are using the Hypertext Transfer Protocol (HTTP).

In simple terms, The World Wide Web is a way of exchanging information between computers on the Internet, tying them together into a vast collection of interactive multimedia resources.

What is HTTP?

HTTP stands for **Hypertext Transfer Protocol**. This is the protocol being used to transfer hypertext documents that makes the World Wide Web possible.

A standard web address such as Yahoo.com is called a URL and here the prefix **http** indicates its protocol.

What is Website?

Website is a collection of various pages written in HTML markup language. This is a location on the web where people can find tutorials on latest technologies. Similarly, there are millions of websites available on the web.

Each page available on the website is called a *web page* and first page of any website is called *home page* for that site.

What is Web Server?

Every Website sits on a computer known as a Web server. This server is always connected to the internet. Every Web server that is connected to the Internet is given a unique address made up of a series of four numbers between 0 and 256 separated by periods. For example, 68.178.157.132 or 68.122.35.127.

When you register a Web address, also known as a domain name, such as tutorialspoint.com you have to specify the IP address of the Web server that will host the site.

What is Web Browser?

Web Browsers are software installed on your PC. To access the Web you need a web browsers, such as Netscape Navigator, Microsoft Internet Explorer or Mozilla Firefox.

On the Web, when you navigate through pages of information through web browser is commonly known as *browsing or surfing*.

What is SMTP Server?

SMTP stands for **S**imple **M**ail **T**ransfer **P**rotocol Server. This server takes care of delivering emails from one server to another server. When you send an email to an email address, it is delivered to its recipient by a SMTP Server.

What is ISP?

ISP stands for **I**nternet **S**ervice **P**rovider. They are the companies who provide you service in terms of internet connection to connect to the Internet.

You will buy space on a Web Server from any Internet Service Provider. This space will be used to host your Website.

What is DNS?

DNS stands for **D**omain **N**ame **S**ystem. When someone types in your domain name, www.example.com, your browser will ask the Domain Name System to find the IP that hosts your site. When you register your domain name, your IP address should be put in a DNS along with your domain name. Without doing it your domain name will not be functioning properly.

What is W3C?

W3C stands for **W**orld **W**ide **W**eb Consortium which is an international consortium of companies involved with the Internet and the Web.

The W3C was founded in 1994 by Tim Berners-Lee, the original architect of the World Wide Web. The organization's purpose is to develop open standards so that the Web evolves in a single direction rather than being splintered among competing factions. The W3C is the chief standards body for HTTP and HTML.

What is Hyperlink?

A hyperlink or simply a link is a selectable element in an electronic document that serves as an access point to other electronic resources. Typically, you click the hyperlink to access the linked resource. Familiar hyperlinks include buttons, icons, image maps, and clickable text links.

What is URL?

URL stands for **U**niform **R**esource **L**ocator, and is used to specify addresses on the World Wide Web. A URL is the fundamental network identification for any resource connected to the web (e.g., hypertext pages, images, and sound files).

A URL will have the following format –

protocol://hostname/other_information

The protocol specifies how information is transferred from a link. The protocol used for web resources is HyperText Transfer Protocol (HTTP). Other protocols compatible with most web browsers include FTP, telnet, newsgroups, and Gopher.

The protocol is followed by a colon, two slashes, and then the domain name. The domain name is the computer on which the resource is located.

Links to particular files or subdirectories may be further specified after the domain name. The directory names are separated by single forward slashes.

Internet Addresses

- ✓ There are two major types of addresses on the Internet. One is a person's e-mail address, and the other is a web page or web site address, which is known as a URL. The proper name for a website address is uniform resource locator (URL).

Example:

Email: india2016@gmail.com

URL: http://localhost:8080/Multifiel.html

Retrieving Data with URL

URL

- ☐ The Uniform Resource Locator was created in 1994 by Tim Berners-Lee.
- ☐ It is used to address a document on the www.
- ☐ The syntax is:-
scheme://host.domain:port/path/filename
- ☐ Example:- http://localhost:8080/Multifiel.html

http://www.w3.org/TR/html4/

scheme *server* *path*

Types of URL

Relative URL:-

- A String is reliable with the URL path syntax is known as relative URL.

Absolute URL:-

- ✓ A complete URL(beginning with a scheme) is known as absolute URL.

Base URL:-

- ✓ Base URL is a URL which is used to convert a relative URL to an absolute URL.

Relative URL

Absolute URL

d/e.html

<http://www.example.org/a/b/c/d/e.html>

../f.html

<http://www.example.org/a/f.html>

../../g.html

<http://www.example.org/g.html>

..h/i.html

<http://www.example.org/a/h/i.html>

/j.html

<http://www.example.org/j.html>

/k/l.html

<http://www.example.org/k/l.html>

URIs, URNs, and IRIs

- ☐ Uniform Resource Identifier (URI), Every URI is defined as consisting of four parts, as follows:
- ☐ <scheme name> : <hierarchical part> [? <query>] [# <fragment>]
- ☐ The **scheme name** consist of a sequence of characters beginning with a letter and followed by any combination of letters, digits, plus ("+"), period ("."), or hyphen ("-").

The **hierarchical part** begins with a double forward slash ("/"), followed by an *authority* part and an optional *path*.

- ☐ The **authority** part holds an optional user-information part, terminated with "@" (e.g. username:password@); a hostname (e.g., domain name or IP address); and an optional port number, preceded by a colon ":".
- ☐ The **path** part is a sequence of segments separated by a forward slash ("/").
- ☐ The **query** is an optional part, separated by a question mark ("?").
- ☐ The **fragment** is an optional part separated from the front parts by a hash ("#").
- ☐ EXAMPLE:
- ☐ http://en.wikipedia.org/wiki/URI#Examples_of_URI_references
- ☐ "http" specifies the 'scheme' name,
- ☐ "en.wikipedia.org" is the 'authority',
- ☐ "/wiki/URI" the 'path' pointing to this article, and "#Examples_of_URI_references" is a 'fragment' pointing to this section.

Markup Languages

- Notation for adding formal structure to text.
- Standard General Markup Language, SGML (1986)
- Markup languages do not *directly* instruct computers like procedural languages rather *indirectly* instruct computer like logical languages.
- They do this by using tags:
 - ✓ elements
 - ✓ attributes
 - ✓ entities
 - ✓ text

PCDATA

- ❑ Parsed Character Data (PCDATA) is a data definition that created in Standard General Markup Language (SGML), and is used also in Extensible Markup Language (XML) .
- ❑ Document Type Definition (DTD) to assign mixed content XML elements.

SGML: a Brief Introduction

- ✓ **Standard General Markup Language**
- ✓ In 1986, ISO approved an international standard for descriptive markup.
- ✓ SGML is a metalanguage(a language that can be used to explain languages) for defining markup languages.
- ✓ HTML is one example of an SGML-defined language

What is the WWW?

- ❑ A hypertext system that runs on top of the Internet, based on Three Main Standards
 - URL
 - HTTP
 - HTML

The Origins of the WWW

- ❑ **WWW was invented by Tim Berners-Lee at CERN(European Organization for Nuclear Research-1989)**
- ❑ Hypertext across the Internet (replacing FTP)
- ❑ Three constituents: HTML + URL + HTTP
- ❑ HTML is an SGML language for hypertext
- ❑ URL is an notation for locating files on servers
- ❑ HTTP is a high-level protocol for file transfers

Hyper Text Transfer Protocol(HTTP)

HTTP Request Methods:

- ✓ HTTP /1.0 Specifies three types of Request Methods

- 1.GET Method.
- 2.POST Method.
- 3.HEAD Method.

HTTP Request Methods:

- ✓ HTTP /1.1 has five additional request Methods.

- 1.OPTIONS
- 2.PUT
- 3.TRACE
- 4.DELETE
- 5.CONNECT

- ☐ The GET Method is used to getting the data from server.
- ☐ The POST Method is used for sending data to the server.
- ☐ When a user wants to know about the headers like MIME(Multipurpose Internet Mail Extensions) Types Like

- 1.Applications
- 2.Audio
- 3.Image
- 4.Message
- 4.Text
- 5.Video

What is HTML?

HTML stands for **H**yper **T**ext **M**arkup **L**anguage. This is the language in which we write web pages for any Website. Even the page you are reading right now is written in HTML.

This is a subset of Standard Generalized Mark-Up Language (SGML) for electronic publishing, the specific standard used for the World Wide Web.

HTML Versions

| Version | Year |
|-----------|------|
| HTML | 1991 |
| HTML+ | 1993 |
| HTML 2.0 | 1995 |
| HTML 3.2 | 1997 |
| HTML 4.01 | 1999 |
| XHTML 1.0 | 2000 |
| HTML5 | 2012 |

Element examples

- ☐ Header of the HTML document:<head>...</head>.
- ☐ Usually the title should be included in the head, for example:

- ☐ <head>
- ☐ <title>The Title</title>
- ☐ </head>
- ☐ Changing Font Size
- ☐ There are three different ways to change the size of the font. Method 1, Method 2, and Method 3.
- ☐ Method 1:
 - ☐ The first method is to use the "big" and "small" tags.
- ☐ Method 2:
 - ☐ You can use + and - along with a number to shrink and grow the font.
- ☐ Method 3:
 - ☐ Headers are abbreviated with the letter "h" and a number from 1 to 6.

Method 1:

- ✓ For example, to increase the font by one degree surround the text with the following tags:
- ✓ <big> chosen text </big>
- ✓ If you want to make it even bigger, you surround it again:
- ✓ <big> <big> chosen text </big> </big>
- ✓ To make the font small, surround it with the small tag:
- ✓ <small> chosen text </small>
- ✓ If you want to make it even smaller, surround it with the tag again:
- ✓ <small> <small> chosen text </small> </small>

Method 2:

- ❖ You can use + and - along with a number to shrink and grow the font:
- ❖ chosen text
- ❖ chosen text (same as <big>)
- ❖ chosen text (plain text)
- ❖ chosen text (same as small>)
- ❖ chosen text

Method3:

- ❖ Headings: HTML headings are defined with the <h1> to <h6> tags:
- ❖ <h1>Heading1</h1>
- ❖ <h2>Heading2</h2>
- ❖ <h3>Heading3</h3>
- ❖ <h4>Heading4</h4>
- ❖ <h5>Heading5</h5>
- ❖ <h6>Heading6</h6>

Paragraph Tags

- ✓ Tag: <p> </p> (Has a closing tag) </> means closed.
- ✓ Attributes:

Align=left, right, center

Code Example:

```
<p align=left>This is a paragraph tag</p>
```

```
<p align=center >This is a paragraph tag</p>
```

What it looks like:

This is a paragraph tag.

This is a paragraph tag.

HTML Text Formatting

- ✓ `<html> <body>`
- ✓ `<p> This text is bold </p>`
- ✓ `<p> This text is strong </p>`
- ✓ `<p> <big>This text is big </big></p>`
- ✓ `<p> <i>This text is italic </i></p>`
- ✓ `<p>This is_{subscript} and ^{superscript}</p>`
- ✓ `</body></html>`
- ✓ `` (bold) tag and also a `` tag since both are displayed in bold type.

Output:

1. **This text is bold**
2. **This text is strong**
3. This text is big
4. *This text is italic*

This is _{subscript} and ^{superscript}

Blinking Text:

- To make text blink surround the chosen text with these tags.
- `<blink> chosen text </blink>`

Html Link Tag:

- HTML links are defined with the `<a>` tag.
- The link address is specified in the **href attribute**:

Example

- `This is a link`
- Href -Hypertext Reference.

Changing Text Color:

- ✓ The "color name" can be one of two things. If you only want to use a common color, you can simply put in the name of that color.
- ✓ ` chosen text `
- ✓ ` Black `
- ✓ ` White (White) `
- ✓ ` Yellow (Yellow) `
- ✓ ` Orange `

- ✓ ` Pink (Pink) `
- ✓ ` Red `
- ✓ ` Green `
- ✓ ` Blue `
- ✓ The second thing you can use as a "color name" is a numeric code for the color.
- ✓ Syntax: ` chosen text `
- ✓ Here is a red color whose hex-code number is `# f f 0 0 0 0 . (# f f 0 0 0 0)`
- ✓ Here is a green color whose hex-code number is-- `# 0 0 f f 0 0 . (# 0 0 f f 0 0)`
- ✓ Here is a blue color whose hex-code number is-- `# 0 0 0 0 f f . (# 0 0 0 0 f f)`
- ✓ Here is a light-blue color whose hex-code number is-- `# 0 0 f f f f . (# 0 0 f f f f)`
- ✓ Here is another blue color whose hex-code number is-- `# 1 f 8 1 a 6 . (# 1 f 8 1 a 6)`

Document Structure

```
<html>
<head><title>My First Web Page</title>
</head>
<body bgcolor="white">
<p>A Paragraph of Text.</p>
</body>
</html>
```

Nesting Tags

- To make something both bold and italic use the following tags:
 - ` <i> chosen text </i> `
- For big and red:
 - ` <big>chosen text </big> `
- Both the color and the size of the font at the same time
 - ` chosen text `

HTML Line Breaks:

- ❖ Use the `
` tag if you want a line break (a new line) without starting a new paragraph:

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

Output

- ❖ This is
a para
graph with line breaks
- ❖ The `
` element is an empty HTML element. It has no end tag.

How To Draw A Line Across The Screen

- ✓ To draw a line across the page type:
- ✓ `<hr>`
- ✓ This stands for "horizontal rule"

Types:

- ✓ `<hr width=75%>`

- ✓ `<hr size=10>`
- ✓ `<hr size=10 noshade>`
- ✓ `<hr noshade>`
- ✓ `<hr width=50% align=left>`
- ✓ `<hr width=50% align=center>`
- ✓ `<hr width=50% align=right>`

How To Center a Section Of Text

- ❖ The center tag also automatically inserts a blank line above and below the chosen text.
- ❖ `<center>chosen text</center>`

How To Indent a Section Of Text

- ❖ The blockquote tag also automatically inserts a blank line above and below the chosen text.
- ❖ `<blockquote>chosen text</blockquote>`
- ❖ `<BLOCKQUOTE ...>` indicates that you are quoting a large section of text.
- ❖ `<Blockquote>` is used to indicate a large portion of referenced text.
- ❖ **HTML List Tags**

| Tag | Description |
|-------------------------|---|
| <code></code> | Defines an ordered list |
| <code></code> | Defines an unordered list |
| <code></code> | Defines a list item |
| <code><dl></code> | Defines a definition list |
| <code><dt></code> | Defines an item in a definition list |
| <code><dd></code> | Defines a description of an item in a definition list |

How To Make unordered Lists

- ❖ An unordered list starts with the `` tag. Each list item starts with the `` tag.
- ❖ ``
 - `Coffee`
 - `Milk```
- ❖ Output:
 - Coffee
 - Milk
- If you want to use a hollow circle, add to the starting tag: **type=circle**, so it looks like `<UL type=circle>`.

- If you want to use a filled square, add to the starting tag: **type=square**, so it looks like
<UL type=square>

How To Make Ordered Lists

- ✓ An ordered list starts with the tag. Each list item starts with the tag.
- ✓
 - Coffee
 - Milk

- ✓ Output
 1. Coffee
 2. Milk
- ✓ It is also possible to change the style of the numbers from 1. 2. 3., to A. B. C., a. b. c., I. II. III., or i. ii. iii.
- ✓ A For capital letters add **type=A** to the starting tag: <OL type=A>.
- ✓ a For lower-case letters add **type=a** to the starting tag: <OL type=a>.
- ✓ I For capital Roman numerals add **type=I** to the starting tag: <OL type=I>.
- ✓ i For lower-case Roman numerals add **type=i** to the starting tag: <OL type=i>.

HTML Definition Lists

- The <dl> tag defines a definition list.
- The <dl> tag is used in combination with <dt> (defines the item in the list) and <dd> (describes the item in the list):
- <dl>
 - <dt>Coffee</dt>
 - <dd>- black hot drink</dd>
 - <dt>Milk</dt>
 - <dd>- white cold drink</dd>
</dl>
- How the HTML code above looks in a browser:
- Coffee - black hot drink
- Milk - white cold drink

Tables

- ✓ <table>...</table>
- ✓ <tr>...</tr> for each row
- ✓ <td>...</td> for each element in a row
- ✓ <th>...</th> for header row

Attributes:

- ✓ align=left, right, center
- ✓ border=x
- ✓ cellpadding=x

- ✓ cellspacing=x
- ✓ width=
- ✓ height=

How these work and look: All these tags must be closed. </>

```
<table border=1 cellpadding=2 cellspacing=2>
<tr>
<td>cell 1</td>
<td>cell 2</td>
</tr>
</table>
```

How it looks:

| | |
|--------|--------|
| Cell 1 | Cell 2 |
|--------|--------|

Table Example

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

| | |
|---------------|---------------|
| row 1, cell 1 | row 1, cell 2 |
| row 2, cell 1 | row 2, cell 2 |

HTML Table Headers

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

```

<th>Header 2</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>

```

Comments

- ✓ <!-- This is a comment -->
- ✓ <!--
This paragraph,
is also a
comment...
-->

Example:

- ✓ <html>
- ✓ <body>
- ✓ <!--This is a comment. Comments are not displayed in the browser-->
- ✓ <p>This is a paragraph.</p>
- ✓ </body>
- ✓ </html>

Output:

?

Html Table Elements:

- ✓ <TABLE>, <CAPTION>,<TR>, <TH>, and <TD>.
- ✓ New elements that provide increased control over table formatting<COL>,
<COLGROUP>, <THEAD>, <TFOOT>, and <TBODY>.

<Caption> Tag:

- ✓ The <caption> tag defines a table caption.
- ✓ The <caption> tag must be inserted immediately after the <table> tag.
- ✓ You can specify only one caption per table.
- ✓ Example:
 - ✓ <html><body>
 - ✓ <table border="1">

- ✓ `<caption>Monthly savings</caption>`
- ✓ `<tr> <th>Month</th> <th>Savings</th> </tr>`
- ✓ `<tr> <td>January</td> <td>$100</td> </tr>`
- ✓ `<tr> <td>February</td> <td>$50</td> </tr>`
- ✓ `</table> </body> </html>`

Monthly savings

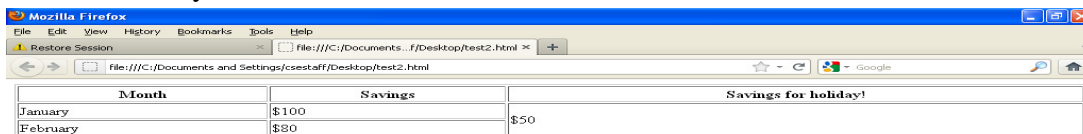
| Month | Savings |
|----------|---------|
| January | \$100 |
| February | \$50 |

✓

ROWSPAN and COLSPAN

- ✓ The rowspan attribute specifies the number of rows a cell should span.
- ✓ **Syntax:** `<td rowspan="number">` .
- ✓ Example:

```
<html><body><table width="100%" border="1">
<tr> <th>Month</th> <th>Savings</th>
<th>Savings for holiday!</th> </tr>
<tr> <td>January</td> <td>$100</td>
<td rowspan="2">$50</td> </tr> <tr>
<td>February</td> <td>$80</td> </tr>
</table> </body></html>
```



The screenshot shows a Mozilla Firefox browser window displaying the rendered HTML table. The table has three columns: 'Month', 'Savings', and 'Savings for holiday!'. The 'Savings for holiday!' column is a single cell that spans two rows, containing the value '\$50'. The 'Savings' column has two rows: 'January' with '\$100' and 'February' with '\$80'.

| Month | Savings | Savings for holiday! |
|----------|---------|----------------------|
| January | \$100 | \$50 |
| February | \$80 | |

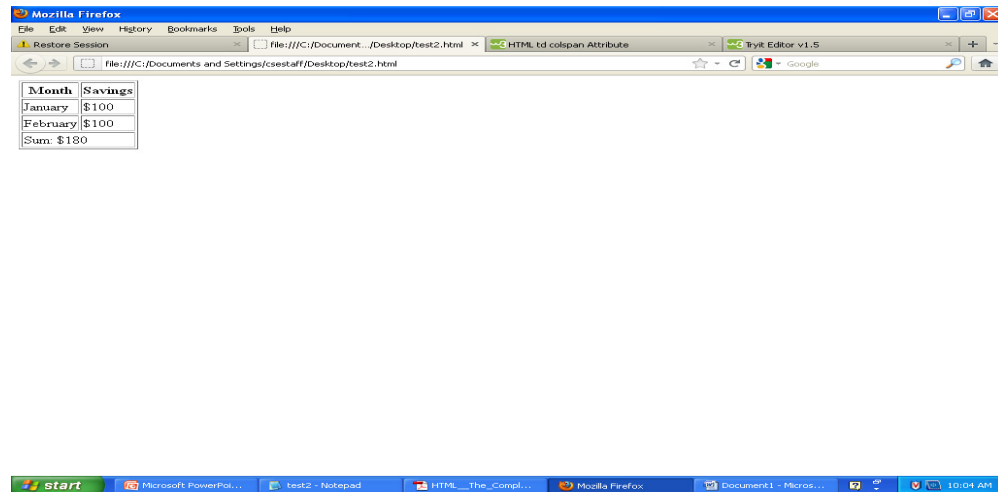


- ✓ The colspan attribute defines the number of columns a cell should span.

✓ **Syntax:** <td colspan="number"> .

✓ Example:

```
<html><body><table border="1">
<tr> <th>Month</th> <th>Savings</th> </tr>
<tr> <td>January</td> <td>$100</td> </tr>
<tr> <td>February</td> <td>$100</td> </tr>
<tr> <td colspan="2">Sum: $180</td> </tr>
</table> </body> </html>
```



HTML <Thead> (Table Head)Tag:

The <thead> tag is used to group header content in an HTML table.

The <thead> element is used in conjunction with the <tbody> and <tfoot> elements to specify each part of a table (header, body, footer).

The <thead> element must have one or more <tr> tags inside.

```
<html><head><style type="text/css">
thead { color:green }
tbody { color:blue;height:50px }
tfoot { color:red }
</style></head><body>
<table border="1">
  <thead> <tr> <th>Month</th> <th>Savings</th> </tr>
</thead>
<tfoot> <tr><td>Sum</td> <td>$180</td> </tr> </tfoot>
<tbody> <tr><td>January</td> <td>$100</td> </tr>
<tr> <td>February</td> <td>$80</td> </tr> </tbody>
</table></body></html>
```

| Month | Savings |
|----------|---------|
| January | \$100 |
| February | \$80 |
| Sum | \$180 |

HTML <Col> Tag:

The <col> tag defines attribute values for one or more columns in a table.

The <col> tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.

The <col> tag can only be used inside a <table> or a <colgroup> element.

In HTML the <col> tag has no end tag.

In XHTML, the <col> tag must be properly closed.

```
<html><body><table border="1">
<colgroup>
  <col span="2" style="background-color:red" />
  <col style="background-color:yellow" />
</colgroup>
<tr> <th>ISBN</th><th>Title</th> <th>Price</th> </tr>
<tr><td>3476896</td> <td>My first HTML</td>
<td>$53</td> </tr>
<tr> <td>5869207</td> <td>My first CSS</td>
<td>$49</td> </tr></table></body></html>
```

| ISBN | Title | Price |
|---------|---------------|-------|
| 3476896 | My first HTML | \$53 |
| 5869207 | My first CSS | \$49 |

HTML <colgroup> Tag

- ✓ The <colgroup> tag is used to group columns in a table for formatting.
- ✓ The <colgroup> tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.
- ✓ The <colgroup> tag can only be used inside a <table> element.

```
<html><body><table width="100%" border="1">
```

```

<colgroup span="2" style="background-color:#FF0000;"></colgroup>
<colgroup style="background-color:#0000FF;"></colgroup>
<tr> <th>ISBN</th><th>Title</th> <th>Price</th> </tr>
<tr> <td>3476896</td> <td>My first HTML</td>
<td>$53</td> </tr>
<tr><td>2489604</td> <td>My first CSS</td> <td>$47</td>
</tr></table></body></html>

```

| ISBN | Title | Price |
|---------|---------------|-------|
| 3476896 | My first HTML | \$53 |
| 2489604 | My first CSS | \$47 |

Special HTML or Common Character Entities

< → <
 > → >
 & → &
 → space
 © → Copyright symbol ©
 ® → Registered trademark
 " → Quotation mark “
 N/A → Trademark TM

Example:

- ✓ <HTML><HEAD>
- ✓ <TITLE>Character Entities Example</TITLE>
- ✓ </HEAD><BODY>
- ✓ <H1 ALIGN="center">Big Company Inc.'s Tagging Products</H1><HR>
- ✓ <P>Character entities like © allow users to insert special characters like ©.</P>
- ✓ <P>One entity that is both useful and abused is the nonbreakingspace.</P>

- ✓ Inserting spaces is easy with

- ✓ Look: S P A
 C E S.

- ✓ <HR><ADDRESS>
- ✓ Contents of this page © 1999 Big Company, Inc.

- ✓ TheWonder Tag <P>™ is a registered trademark of Big Company, Inc.
- ✓ </ADDRESS></BODY></HTML>



Address Tag:

- ✓ The <address> tag defines the contact information for the author or owner of a document. This way, the reader is able to contact the document's owner.
- ✓ The <address> element is usually added to the header or footer of a webpage.

<address>

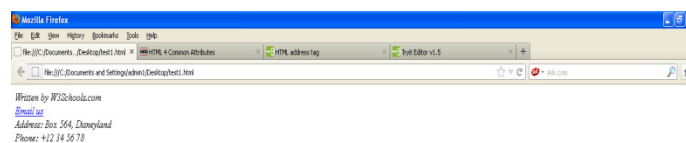
Written by W3Schools.com

Email us

Address: Box 564, Disneyland

Phone: +12 34 56 78

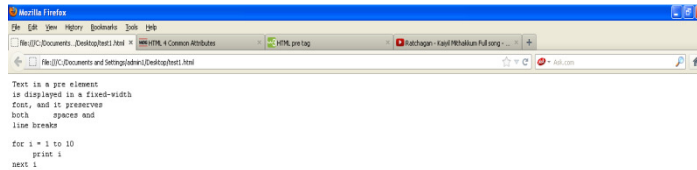
</address>



The <pre> tag defines preformatted text:

- ✓ If you want the text to appear exactly as you type it, use these tag, and it preserves both spaces and line breaks.
- ✓ Syntax:<pre> text</pre>

<pre>Text in a pre element
is displayed in a fixed-width
font, and it preserves
both spaces and
line breaks</pre>
<pre>for i = 1 to 10
 print i
next I </pre>



Physical Text-Formatting Elements:

<I>... </I> Italics
... Bold
<TT>... </TT> Typewriter (monospaced)
<U>... </U> Underline
<STRIKE> ... </STRIKE> Strikethrough
<S> ... </S> Alternative element form of strikethrough
_{...} Subscript
^{...} Superscript
<BIG>... </BIG> Bigger font (one font size bigger)
<SMALL> ... </SMALL> Smaller font (one font size smaller)

Logical Text-Formatting Elements:

<ABBR>... </ABBR> Abbreviation
<CITE>... </CITE> Citation
<CODE> ... </CODE> Source code
<DFN> ... </DFN> Definition
 ... Emphasis
<KBD>... </KBD> Keystrokes
<SAMP> ... </SAMP> Sample (example information)
 ... Strong emphasis
<VAR>... </VAR> Programming variable

HTML 4 Common Attributes

- These attributes are divided into core attributes, internationalization attributes, and scripting events.
- HTML 4's core attributes are
 - ID, CLASS, STYLE, and TITLE.
- HTML 4's internationalization attributes are
 - LANG, DIR
- HTML 4's scripting events are
 - ONCLICK, ONDBLCLICK, ONMOUSEDOWN,
 - ONMOUSEUP, ONMOUSEOVER, etc..
- The **ID** attribute is used to set a unique name for a tag in a document.
- No two elements can have the same **ID** value in a single document.
- The attribute's value must begin with a letter in the range A-Z or a-z and may be followed by letters (A-Za-z), digits (0-9), hyphens ("-"), underscores ("_"), colons (":"), and periods (".").
- The value is case-sensitive.
- <P ID="First Paragraph">This is the first paragraph of text.</P>
- <P ID="Second Paragraph">This is the second paragraph of text.</P>
- ✓ The **CLASS** attribute is used to indicate the class or classes that a tag may belong to.
- ✓ The **CLASS** attribute specifies the element to be a member of one or more classes.
- ✓ <P ID="FirstParagraph" CLASS="important">
This is the first paragraph of text. </P>.

STYLE attribute

The **STYLE** attribute is used to add style sheet information directly to a tag.

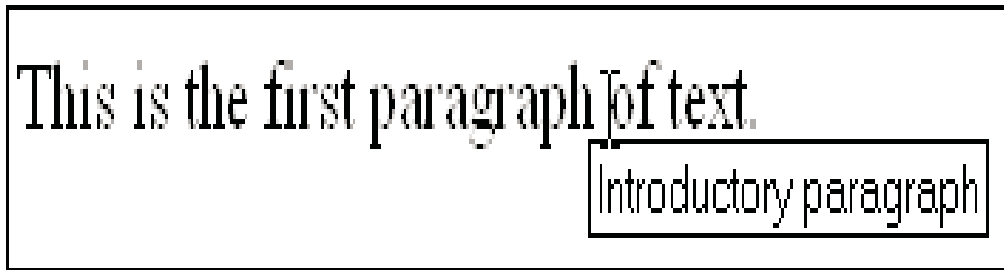
<P STYLE="font-size: 18pt">First paragraph text.</P>

The **TITLE** is used to provide advisory text about a tag or its contents.

<P TITLE="Introductory paragraph">

This is the first paragraph of text.</P>

Output:



HTML 4's internationalization attributes:

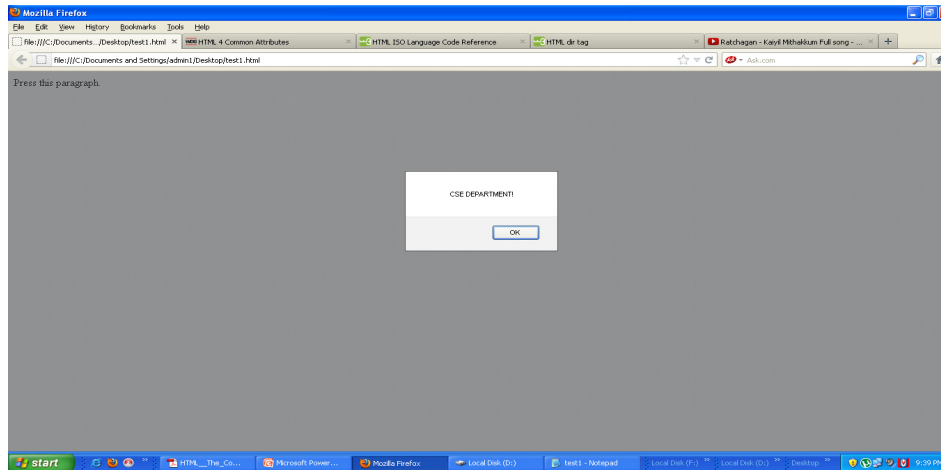
- ✓ The HTML Lang attribute can be used to declare the language of a Web page or a portion of a Web page.
- ✓ `<html lang="en"> </html>`.
- ✓ The HTML DIR Specifies the text direction for the content in an element.
- ✓ The values are LTR (left to right) or RTL (right to left).
- ✓ `<P DIR="RTL">This is a right-to-left paragraph.</P>`
- ✓ This is a right-to-left paragraph.
- ✓ `<P DIR="LTR">This is a left-to-right paragraph.</P>`
- ✓ This is a left-to-right paragraph

HTML 4's Common scripting events:

- ✓ ONCLICK, when the mouse button is clicked on an element;
- ✓ ONDBLCLICK, when the mouse button is double-clicked on an element;
- ✓ ONMOUSEDOWN, when the mouse button is pressed over an element;
- ✓ ONMOUSEUP, when the mouse button is released over an element;
- ✓ ONMOUSEOVER, when the mouse is moved onto an element;
- ✓ ONMOUSEMOVE, when the mouse is moved while over an element;
- ✓ ONMOUSEOUT, when the mouse is moved away from an element;
- ✓ ONKEYPRESS, when a key is pressed and released over an element;
- ✓ ONKEYDOWN, when a key is pressed down over an element;
- ✓ ONKEYUP, when a key is released over an element.

Example:

```
<html>
<head>
<P onclick="alert('cse department!')">Press this paragraph.</P>
</head>
</html>
```

HTML Block Elements & inline Elements

- Block level elements normally start (and end) with a new line when displayed in a browser.
- HTML block level elements can appear in the body of an HTML page.
- Block level elements create larger structures .
- List of block level elements
- P, h1, h2, h3, h4, h5, h6, ol, ul,
- Blockquote, dl, div, form, hr, table
- Inline elements are normally displayed without starting a new line.
- HTML inline level elements can appear in the body of an HTML page.
- inline elements create shorter structures.
- List of inline elements
- i, b , a href, strong
- span, sub, sup
- a, br, img, map

HTML Grouping Tags

| Tag | Description |
|--------|----------------|
| <div> | Defines a div |
| | Defines a span |

- The SPAN and DIV elements are very useful when dealing with Cascading Style Sheets.
- The HTML<div> tag defines a division or a section in an HTML document.
- The HTML tag is used for grouping and applying styles to inline elements.

The HTML <div> Element

- ❖ The HTML <div> element is a block level element that can be used as a **container for grouping other HTML elements.**

- ❖ When used together with CSS, the <div> element can be used to set style attributes to large blocks of content.

```
<html><body>
<h3>This is a header</h3>
<p>This is a paragraph.</p>
<div style="color:Red">
  <h3>This is a header</h3>
  <p>This is a paragraph.</p>
</div></body></html>
```

This is a header

This is a paragraph.

This is a header

This is a paragraph.

The HTML Element

- ❖ The HTML element is an inline element that can be used as a **container for text**.
- ❖ When used together with CSS, the element can be used to set style attributes to parts of the text.
- ❖ The tag can be inside <p> tags or <div> tags.

```
<html><body>
<p>My mother has <span style="color:lightblue;font-weight:bold">light blue</span>
eyes and my father has <span style="color:darkolivegreen;font-weight:bold">dark green</span>
eyes.</p>
</body></html>
```

My mother has light blue eyes and my father has dark green eyes.

Forms and Its Objects

Control Types

HTML Defines the following control types:

- Buttons
 - Authors may create three types of buttons:
 - Submit Buttons

- Reset Buttons
- Push Buttons
- Checkboxes
- Radio Buttons
- Menus
- Text input

HTML Forms

- ✓ HTML Forms are used to select different kinds of user input.
- ✓ HTML forms are used to pass data to a server.
- ✓ A form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more.
- ✓ A form can also contain select lists, textarea, fieldset, legend, and label elements.
- ✓ The <form> tag is used to create an HTML form:
- ✓ <form>

.
input elements

.
</form>

Attributes:

- Type - This attribute specifies the type of control to create. The default type is text
- Name - This attribute assigns the control name.
- Value - This attribute specifies the initial value of the control.
- Size - This attribute tells the initial width of the control.

Control Type created with INPUT

Text Fields

- ❖ Syntax: <input type="text" /> defines a one-line input field that a user can enter text.

<form>

First name: <input type="text" name="firstname" />

Last name: <input type="text" name="lastname" />

</form>

Output: First name: Last name:

A text field:

<input type="text" name="textfield" value="with an initial value" />

A text field:

Password Field

- ❖ Syntax: `<input type="password" />` defines a password field:
- ❖ `<form>Password: <input type="password" name="pwd" />`
`</form>`
Password:
- ❖ A password field:
`<input type="password" name="textfield3" value="secret" />`

A password field:

Radio Buttons

- `<input type="radio" />` defines a radio button.
- Radio buttons let a user select ONLY ONE of a limited number of choices:

`<form>`

`<input type="radio" name="gender" value="male" /> Male`

`
<input type="radio" name="gender" value="female" /> Female</form>`

- Male
- Female

Checkboxes

- ✓ `<input type="checkbox" />` defines a checkbox.
- ✓ Checkboxes let a user select ONE or MORE options of a limited number of choices.
- ✓ `<form>`

`<input type="checkbox" name="vehicle" value="Bike" /> I have a bike
`

`<input type="checkbox" name="vehicle" value="Car" /> I have a car`

`</form>`

☐ I have a bike

☐ I have a car

Submit Button

- `<input type="submit" />` defines a submit button.
- A submit button is used to send form data to a server.
- `<form > Username: <input type="text" name="user" />`
`<input type="submit" value="Submit" /></form>`

Username:

- Push Button:
- The `<button>` tag defines a push button:

`<button type="button">Click Me!</button>`.

Click Me!

Field set & legend:

- ✓ The <fieldset> tag is used to group related elements in a form.
- ✓ The <fieldset> tag draws a box around the related elements.
- ✓ The <legend> tag defines a caption for the <fieldset> element.

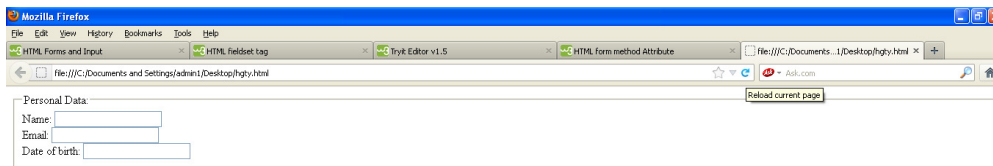
```
<form><fieldset><legend>Personal Data:</legend>
```

```
Name: <input type="text" /><br />
```

```
Email: <input type="text" /><br />
```

```
Date of birth: <input type="text" />
```

```
</fieldset> </form>
```



The Label Element:

- ❖ The label element is used to combine text with another element of the form.
- ❖ Syntax: <label>Label name:-Any control type </label>
- ❖ Example:- <form>

```
<label name="male">Male</label>
```

```
<input type="radio" name="sex" id="male" />
```

```
<br />
```

```
<label name="female">Female</label>
```

```
<input type="radio" name="sex" id="female" />
```

```
</form>
```

- Male
- Female

The Text area Element:

- ✓ The Text area is allows the user to enter multiple lines of data.
- ✓ Textarea Vs Input
 - ✓ Input is an empty tag.
 - ✓ Textarea is not empty – any character data placed between the start and end tags of a textarea is displayed as default text in the textarea box.
- ✓ A multi-line text field :
- ✓ `<textarea name="textarea" cols="24" rows="2">Hello</textarea>`

A multi-line text field



- ✓ will select a single option.
- ✓ It Create a drop-down list.
- ✓ A menu or list:


```
<select name="select">
  <option value="red">red</option>
  <option value="green">green</option>
  <option value="BLUE">blue</option>
</select>
```

A menu or list:



Design a Form

First name:

Last name:

Password:

Sex: ☐ Male ☐ Female

Vehicle: ☐ I have a bike ☐ I have a car

Select a Car:

```
<html><head><title>My First Web Page</title>
</head><form>
First name: <input type="text" name="firstname" /><br />
Last name: <input type="text" name="lastname" /> <br />
Password: <input type="password" name="pwd" /><br />
Sex:<input type="radio" name="sex" value="male" /> Male
<input type="radio" name="sex" value="female" /> Female<br />
Vehicle:<input type="checkbox" name="vehicle" value="Bike" /> I have a bike
<input type="checkbox" name="vehicle" value="Car" /> I have a car<br />
Select a Car:<select>
<option value="volvo">Volvo</option>
<option value="i10">i10</option>
<option value="i20">i20</option>
<option value="santro">santro</option>
</select>
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Image Maps

- ❑ ``
- ❑ JPEG
 - Best for photos
 - Public standard
- ❑ GIF
 - Best for simple images
 - Older standard
- ❑ PNG – Portable Network Graphics
 - Public standard replacement for GIF
 - a bitmap image file format(.png)
- ❑ SVG – Scalable Vector Graphics(.svg)
 - Series of drawing commands
 - Uses XML
- ❑ `img src="URL of image file">`
- ❑ JPEG
 - Best for photos
 - Public standard
- ❑ GIF
 - Best for simple images
 - Older standard
- ❑ PNG – Portable Network Graphics
 - Public standard replacement for GIF
 - a bitmap image file format(.png)
- ❑ SVG – Scalable Vector Graphics(.svg)
 - Series of drawing commands
 - Uses XML

Syntax for defining an image:

- ✓ ``

Example:

- ✓ ``

HTML Images - Set Height and Width of an Image

- ✓ ``

Example:

- ✓ `<html><body>`
- ✓ `<h2>Dept of CSE</h2>`
- ✓ ``
- ✓ `</body></html>`

HTML Hyperlinks (Links)

- ❖ The HTML code for a link is simple. It looks like this:
- ❖ `Link text`

Example:

- ❖ `Visit W3Schools` - which will display like this:
- ❖ Visit W3Schools.

HTML Links - The target Attribute

- ❖ The target attribute specifies where to open the linked document.
- ❖ `Visit W3Schools!`

Making an Image Also Be A Link

- You can make a picture *also* be a link to a web site by combining the links for each.
- The link tag (`a href=`) goes around the outside, with the image tag (`img src=`) taking the place of the words of the link.
- Syntax:

```
<a href="http://www.p4.com/janeinfo/janeinfo.html">
```

- ``



Or, with text too:

Syntax:

```
<a href="http://www.p4.com/janeinfo/janeinfo.html">
```

```
Jane Austen Information
```

```
Page</a>
```



[Jane Austen Information Page](#)

Meaning Of These Tags

1. If you want to use a graphic (.jpg or .gif) as your background use a tag like:`<body background="http://www.spring.net/~anneh/back.jpg">`
2. If you want all of the text of the page to appear blue you would use the body tag:
`<body text="#0000ff">`
3. If you want the unfollowed links to be a certain color, like green, then use the tag:
`<body link="#00ff00">`

4. If you want the followed links to be a certain color, like red, then use the tag:
`<body vlink="#ff0000">`

Image Maps

- ✓ Image maps are images with clickable areas that usually link to another page.
- ✓ **Types:**
- ✓ The two basic types of image maps are
 - server-side image maps and
 - client-side image maps.

Server-Side Image Maps

- ✓ To specify a server-side image map, you use the `<A>` element to enclose a specially marked `` element.
- ✓ The `<A>` element `HREF` attribute should be set to the URL of a program or map file to decode the image map.
- ✓ The `` element must contain the attribute `ISMAP` so that the browser can decode the image appropriately.
- ✓ The `ismap` attribute specifies that the image is part of a server-side image-map (an image-map is an image with clickable areas).
- ✓ When clicking on a server-side image-map, the click coordinates are sent to the server as a URL query string.
- ✓ Syntax: ``
- ✓ `<HTML><HEAD><TITLE>Server-side Image Map`
Example `</TITLE></HEAD><BODY>`

```
<H1 ALIGN="center">Server-side Imagemap Test</H1>
```

```
<DIV ALIGN="center">
```

```
<A HREF="shapes.map">
```

```
<IMG SRC="shapes.gif" ISMAP BORDER="40" WIDTH="400" HEIGHT="200"></A>
```

```
</DIV> </BODY>
```

```
</HTML>
```

Server-side Imagemap Test



Client-Side Image Maps

- ✓ Client-side image map is to add the USEMAP attribute to the element and have it reference a <MAP> element that defines the image map's active areas.
- ✓ The usemap attribute specifies an image as a client-side image-map.
- ✓ An image-map is an image with clickable areas.
- ✓ The usemap attribute is associated with a <map> element's name or id attribute, and creates a relationship between the image and the map.
- ✓ Syntax:

HTML <area> tag:

- ✓ The <area> tag defines an area inside an image-map.
- ✓ The <area> element is always nested inside a <map> tag.
- ✓ The usemap attribute in the tag is associated with the <map> element's name attribute, and creates a relationship between the image and the map.

Optional Attributes:

- ✓ Coords(*coordinates*):- Specifies the coordinates of an area.
 - rect: left, top, right, bottom
 - circle: center-x, center-y, radius
 - poly: x1, y1, x2, y2, ...
- ✓ href :- Specifies the hyperlink target for the area STF .
- ✓ nohref:- nohref Specifies that an area has no associated link.
- ✓ Shape :- default,rect,circle,poly -Specifies the shape of an area.
- ✓ Target:-blank,parent,self,top -Specifies where to open the linked page specified in the href attribute

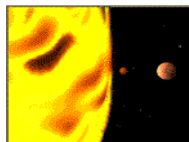
Example

```
<html><body> <p>Click on the sun or on one of the planets to watch it closer:</p>

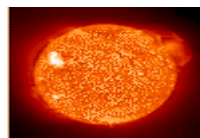
<map name="planetmap">
<area shape="rect" coords="0,0,82,126" alt="Sun" href="sun.htm" />
<area shape="circle" coords="90,58,3" alt="Mercury" href="mercur.htm" />
<area shape="circle" coords="124,58,8" alt="Venus" href="venus.htm" />
</map></body></html>
```

Output:

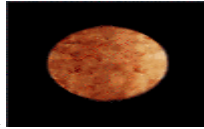
Planet Map



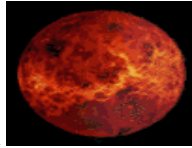
Sun.html



Merglobe.html



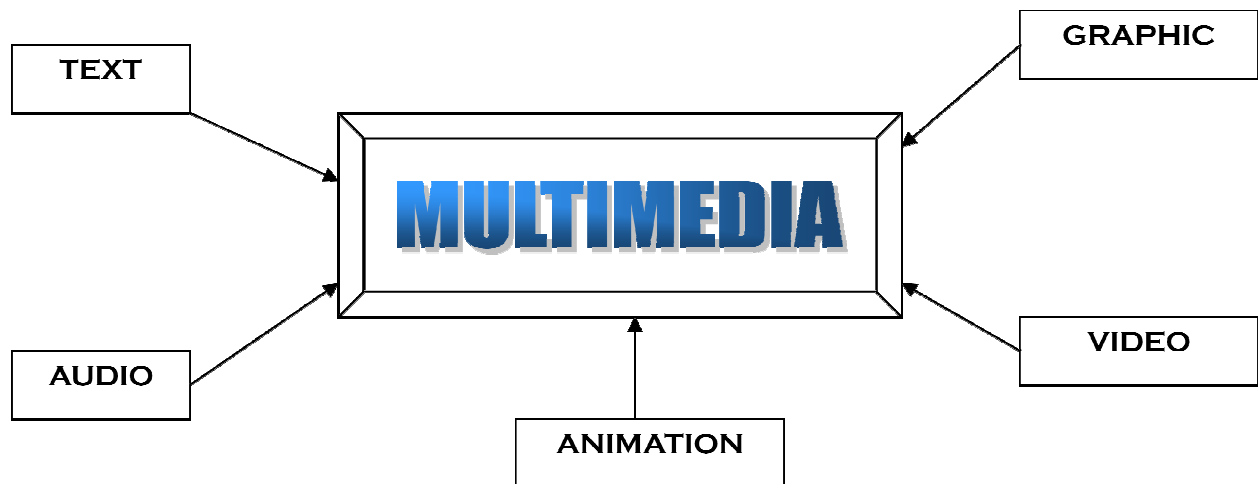
Venglobe.html



Multimedia Components

Definition of Multimedia

Multimedia is a combination of text, graphic, sound, animation, and video that is delivered interactively to the user by electronic or digitally manipulated.



HTML Multimedia Components



Video Formats

| Format | File | Description |
|------------|---------------|---|
| AVI | .avi | <ul style="list-style-type: none"> ✓ The AVI (Audio Video Interleave) format was developed by Microsoft. ✓ The AVI format is supported by all computers running Windows, and by all the most popular web browsers. ✓ It is a very common format on the Internet, but not always possible to play on non-Windows computers. |
| WMV | .wmv | <ul style="list-style-type: none"> ➤ The Windows Media format is developed by Microsoft. ➤ Windows Media is a common format on the Internet, but Windows Media movies cannot be played on non-Windows computer without an extra (free) component installed. |
| MPEG | .mpg .mpeg | <ul style="list-style-type: none"> ✓ The MPEG (Moving Pictures Expert Group) format is the most popular format on the Internet. ✓ It is cross-platform, and supported by all the most popular web browsers. |
| Quick Time | .mov | <ul style="list-style-type: none"> ✓ The QuickTime format is developed by Apple. ✓ QuickTime is a common format on the Internet, but QuickTime movies cannot be played on a Windows computer without an extra (free) component installed. |
| Real Video | .rm .ram | <ul style="list-style-type: none"> ✓ The RealVideo format was developed for the Internet by Real Media. ✓ The format allows streaming of video (on-line video, Internet TV) with low bandwidths. |
| Flash | .swf .flv | <ul style="list-style-type: none"> ✓ The Flash (Shockwave) format was developed by Macromedia |
| Mpeg-4 | .mp4 | <ul style="list-style-type: none"> ✓ Mpeg-4 (with H.264 video compression) is the new format for the internet. |

Sound Formats

| Format | File | Description |
|------------|---------------|---|
| Real Audio | .rm .ram | ✓ The RealAudio format was developed for the Internet by Real Media. ✓ The format also supports video. |
| Wave | .wav | ✓ The Wave (waveform) format is developed by IBM and Microsoft. ✓ It is supported by all computers running Windows, and by all the most popular web browsers (except Google Chrome). |
| WMA | .wma | ✓ The WMA format (Windows Media Audio), compares in quality to MP3. ✓ It is compatible with most players, except the iPod. |
| MP3 | .mp3 .mpga | ✓ MP3 files are actually the sound part of MPEG files. ✓ The MPEG format was originally developed for video by the Moving Pictures Experts Group. ✓ MP3 is one of the most popular sound formats for music. |
| MIDI | .mid .midi | ✓ The MIDI (Musical Instrument Digital Interface) is a format for electronic music devices like PC sound card. |

Multimedia Components

- ✓ The easiest way to add video or sound to your web site is to include the special HTML tag called <embed>.
- ✓ This tag instruct the browser itself to include controls for the multimedia automatically.
- ✓ You should use <no embed> tag along with this tag to handle browsers who do not support embed tag.
- ✓ Syntax:

<EMBED SRC="welcome.avi" HEIGHT="100" WIDTH="100">

<NOEMBED>

 </NOEMBED>

- ✓ **Attributes:** Following is the list of important attributes for <embed> element.
- ✓ **align** - It takes either *center*, *left* or *right*.
- ✓ **auto start** - Indicates if the media should start automatically. Netscape default is true, Internet Explorer is false.
- ✓ **loop** - Specifies if the sound should be played continuously (set loop to true), a certain number of times (a positive value) or not at all (false). This is supported by Netscape only.

- ✓ **play count** - Specifies the number of times to play the sound. This is alternat option for *loop* if you are usiong IE.
- ✓ **hidden** - Defines if the object shows on the page. A false value means no and true means yes.
- ✓ **height** - Height of the object in pixels or en.
- ✓ **width** - Width of the object in pixels or en.
- ✓ **pluginspage** - Specifies the URL to get the plugin software.
- ✓ **name** - A name used to reference the object.
- ✓ **src** - URL of the object to be embedded. This can be any recognizable by the user's browser. It could be .mid, .wav, .mp3, .avi and so on).
- ✓ **volume** - Controls volume of the sound. Can be from 0 (off) to 100 (full volume). This attribute is supported by Netscape only.

HTML - Video Media Types

Flash movies (.swf), AVI's (.avi), and MOV's (.mov) file types are supported by the embed tag.

- ✓ .swf files - are the file types created by Macromedia's Flash program.
- ✓ .wmv files - are Microsoft's Window's Media Video file types.
- ✓ .mov files - are Apple's Quick Time Movie format.
- ✓ .mpeg files - are movie files created by the Moving Pictures Expert Group.

Background Audio - The <bgsound> Element:

- You can use the <bgsound> tag to play a soundtrack in the background.
- This tag is for Internet Explorer documents only.
- Sounds can either be samples (WAV or AU format) or MIDI format.
- Syntax:<BGSOUND SRC="start.wav">

```
<bgsound src="/html/yourfile.mid" > <noembed></noembed> </bgsound>
```

| Attribute | Explanation | Example | Source |
|-------------------|--|--|----------|
| SRC=URL | Specifies the address of a sound to be displayed. | <BGSOUND SRC="boing.wav"> | IExplore |
| LOOP=n | Specifies how many times a sound will loop when activated. | <BGSOUND SRC="boing.wav"LOOP=5> | IExplore |
| LOOP=INF INITE | If n=-1, or if LOOP=INFINITE is specified,it will loop indefinitely. | <BGSOUND SRC="boing.wav" OOP=INFINITE> | IExplore |

HTML Audio Sounds

- ✓ Displaying audio in HTML is not easy!
- ✓ You must add a lot of tricks to make sure your audio files will play in all browsers (Internet Explorer, Chrome, Firefox, Safari, Opera) and on all hardware (PC, Mac , iPad, iPhone).

Frames

HTML Frames

- ✓ Frames allow you to divide the page into several rectangular areas and to display a separate document in each rectangle.
- ✓ Each of those rectangles is called a "frame".

Example:

```
<html> <body>
<frameset cols="30%,*">
  <frame src="menu.html">
  <frame src="content.html">
</frameset> </body> </html>
```

Attributes:

- ✓ **frameset** - The parent tag that defines the characteristics of this frames page. Individual frames are defined inside it.
- ✓ **frameset cols="#%, *"** - The width that each frame will have. In the above example, we chose the menu (the 1st column) to be 30% of the total page and used a "*", which means the content (the 2nd column) will use the remaining width for itself (70%).
- ✓ **frame src=""** - The URL of the web page to load into the frame.

Adding a Banner or Title Frame

Add a row to the top for a title and graphics with the code as follows:

```
<html> <body>
<frameset rows="20%,*">
  <frame src="title.html">
  <frameset cols="30%,*">
    <frame src="menu.html">
    <frame src="content.html">
  </frameset>
</frameset> </body> </html>
```

- ✓ *Framespacing* and *border* are the same attribute, but some browsers only recognize one or the other, so use both, with the same value, to be safe.
- ✓ **frameborder="#"** - Determines whether there will be a border.
- ✓ **border="#"** - Modifies the border width.

- ✓ framespacing="#" -Modifies the border width, used by Internet Explorer.

Example of the same frameset without the borders

```
<html> <body>
<frameset border="0" frameborder="0"          framespacing="0" rows="20%,*">
  <frame src="title.html">
<frameset border="0" frameborder="0"  framespacing="0" cols="30%,*">
  <frame src="menu.html">
  <frame src="content.html">
</frameset> </frameset> </body> </html>
```

Two Column Frameset

The frameset (frame_example_frameset_1.html):

```
<html> <head> <title>Frameset page</title> </head> <frameset cols = "25%, *">
<frame src = "frame_example_left.html" />
<frame src = "frame_example_right.html" />
</frameset> </html>
```

The left frame (frame_example_left.html):

```
<html> <body style="background-color:green"> <p>This is the left frame
(frame_example_left.html).</p> </body> </html>
```

The right frame (frame_example_right.html):

```
<html> <body style="background-color:yellow"> <p>This is the right frame
(frame_example_right.html).</p> </body> </html>
```

DHTML

- ❖ DHTML = Dynamic HTML
- ❖ It allows you to build rich client interfaces and to modify them dynamically
- ❖ There is no DHTML standard
- ❖ It is not a W3C, IEEE, ISO or anything else standard
- ❖ DHTML is a collection of several standards
- ❖ DHTML consists of HTML/XHTML, CSS, DOM and
- ❖ JavaScript (or ECMAScript)

Differences Between HTML and XHTML

- ✓ HTML tags and attributes must be lowercase
- ✓ All attribute values must be quoted
- ✓ All elements that can contain others require end tags
- ✓ Empty elements must either have an end tag or self-close
- ✓ All attributes must be name/value pairs
- ✓ The name attribute is deprecated. Use id instead.

Style sheets

What is CSS

- ✓ CSS stands for Cascading Style Sheets.
- ✓ Styles define how to display HTML elements.
- ✓ Styles are normally stored in Style Sheets.
- ✓ Styles were added to HTML 4.0 to solve a problem. (Netscape vs MS Internet Explorer)
- ✓ External Style Sheets can save you a lot of work.
- ✓ External Style Sheets are stored in CSS files.
- ✓ Multiple style definitions will cascade into one. priority:
 - Inline Style (inside HTML element)
 - Internal Style Sheet (inside the <head> tag)
 - External Style Sheet
 - Browser default style

Styling HTML with CSS:

CSS stands for **C**ascading **S**tyle **S**heets

Styling can be added to HTML elements in 3 ways:

- Inline - using a **style attribute** in HTML elements
- Internal - using a **<style> element** in the HTML <head> section
- External - using one or more **external CSS files**

Inline Styling (Inline CSS)

Inline styling is used to apply a unique style to a single HTML element:

Inline styling uses the **style** attribute.

This example changes the text color of the <h1> element to blue:

Example:

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

Internal Styling (Internal CSS)

Internal styling is used to define a style for one HTML page.

Internal styling is defined in the **<head>** section of an HTML page, within a **<style>** element:

Example:

```
<!DOCTYPE html>
<html>
<head>
```

```
<style>
body {background-color:lightgrey;}
h1 {color:blue;}
p {color:green;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External Styling (External CSS)

An external style sheet is used to define the style for many pages.

With an **external style sheet**, you can change the look of an entire web site by changing one file!

To use an external style sheet, add a link to it in the **<head>** section of the HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

- Use the HTML **style** attribute for inline styling
- Use the HTML **<style>** element to define internal CSS
- Use the HTML **<link>** element to refer to an external CSS file
- Use the HTML **<head>** element to store **<style>** and **<link>** elements
- Use the CSS **color** property for text colors
- Use the CSS **font-family** property for text fonts
- Use the CSS **font-size** property for text sizes
- Use the CSS **border** property for visible element borders

- Use the CSS **padding** property for space inside the border
- Use the CSS **margin** property for space outside the border

HTML Style Tags

✓ Style

Link(HTML Images)

HTML Links - Colors

When you move the mouse over a link, two things will normally happen:

- The mouse arrow will turn into a little hand
- The color of the link element will change

By default, a link will appear like this (in all browsers):

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

CSS Fonts

The CSS **color** property defines the text color to be used for the HTML element.

The CSS **font-family** property defines the font to be used for the HTML element.

The CSS **font-size** property defines the text size to be used for the HTML element.

Cascading Order

What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style (inside a specific HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or a browser default value.

References:

1. Thomas A. Powell, "HTML: The Complete Reference", Osborne/Mc Graw Hill, Third Edition, 2001.
2. <http://www.w3schools.com>
3. <http://www.tutorialspoint.com/>

SCSX1016 Web Technology

UNIT-II

XML

What is XML?

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML Example

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

XML is a software- and hardware-independent tool for storing and transporting data.

Why Study XML?

- XML plays an important role in many IT systems.
- For this reason, it is important for all software developers to have a good understanding of XML.
- Before you continue, you should also have a basic understanding of:
 - ✓ HTML
 - ✓ JavaScript

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Simplifies Things

- It simplifies data sharing

- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

Advantages of using XML.

- It is as easy as HTML.
- XML is fully compatible with applications like JAVA, and it can be combined with any application which is capable of processing XML irrespective of the platform it is being used on.
- XML is an extremely portable language to the extent that it can be used on large networks with multiple platforms like the internet, and it can be used on handhelds or palmtops or PDAs.
- XML is an extendable language, meaning that you can create your own tags, or use the tags which have already been created.

There are other advantages of using XML.

- It is a platform independent language.
- It can be deployed on any network if it is amicable for usage with the application in use.

If the application can work along with XML, then XML can work on any platform and has no boundaries.

XML Applications

database interchange:

– example: home health care in the US (data interchange between hospitals and health agencies)
 current: log into hospital, see records in browser, print them and key them into own database
 XML: log into hospital, drag records onto own database

- present different web views to clients
- tailored information discovery

distributed processing:

- scheduling applications: airlines, trains, buses, subways, restaurants, movies, plays, concerts, ...
- commercial applications: shopping educational applications: online help customer-support applications: lawn-mower maintenance to support for computers
- view selection: switch between views without downloading data again dynamic TOC without data reload switching between languages sorting phone books

web agents:

- intelligent searches over the web search criteria and searched documents have to be expressed in standard format (e.g. XML); structural requirements beyond HTML;
- example: 500-channel cable TV and personalised TV guide across entire spectrum of providers user preferences and program description in XML

Benefits of XML

- **Simplicity**
Information coded in XML is easy to read and understand, plus it can be processed easily by computers.
- **Openness**
XML is a W3C standard, endorsed by software industry market leaders.
- **Extensibility**
There is no fixed set of tags.
New tags can be created as they are needed.
- **Self-description**
XML documents can be stored without [schemas] because they contain meta data; any XML tag can possess an unlimited number of attributes such as author or version.
- **Contains machine-readable context information**
Tags, attributes and element structure provide context information ... opening up new possibilities for highly efficient search engines, intelligent data mining, agents, etc.
- **Separates content from presentation**
XML tags describe meaning not presentation.
The look and feel of an XML document can be controlled by XSL stylesheets, allowing the look of a document (or of a complete Web site) to be changed without touching the content of the document.
Multiple views or presentations of the same content are easily rendered.
- **Supports multilingual documents and Unicode**
This is important for the internationalization of applications.
- **Facilitates the comparison and aggregation of data**
The tree structure of XML documents allows documents to be compared and aggregated efficiently element by element.
- **Can embed multiple data types**
XML documents can contain any possible data type — from multimedia data (image, sound, video) to active components (Java applets, ActiveX).
- **Can embed existing data**
Mapping existing data structures like file systems or relational databases to XML is simple....
- **Provides a “one-server view” for distributed data**
XML documents can consist of nested elements that are distributed over multiple remote servers. XML is currently the most sophisticated format for distributed data — the World Wide Web can be seen as one huge XML database.
- **Rapid adoption by industry**
Software AG, IBM, Sun, Microsoft, Netscape, DataChannel, SAP ...

Advantages of XML over HTML

The root cause of the problem lies in **HTML (Hyper Text Markup Language)**, the defacto standard for web publication. The **major problem with HTML** is its 'fixed tagset'. This tagset is mainly for display of the content and HTML provides no tag to address the content precisely.

XML (extensible Markup Language) designed by W3C (World Wide Web Consortium) promises a possible solution to this problem.

The **major advantage of XML over HTML** is its extensibility i.e., provision of user defined tags and attributes to identify the structural elements of a document. XML also provides structural complexity to define document structure that can be nested at any level of complexity.

XML also facilitates the transfer of structured data between servers. XML describes data, such as city name, temperature and barometric pressure, while HTML defines tags that describe how the data should be displayed, such as with a bulleted list or a table.

| HTML | XML |
|--|---|
| <pre><html> <title>Course Roster</title> <body> <center> <h1>Course Roster</h1> <h2>XML Programming</h2> <h3>Department: EECS</h3> <p> <table border=2> <tr> <th>Teacher</th> <td>Paul Thompson</td> </tr><tr> <th>Student List</th> <td>Ron Jones Uma Abingdon Lindsay Garmon </td> </tr> </table> </center> </body> </html></pre> | <pre><?xml version="1.0"?> <course> <name>Java Programming</name> <department>EECS</department> <teacher> <name>Paul Thompson</name> </teacher> <student> <name>Ron Jones</name> </student> <student> <name>Uma Abingdon</name> </student> <student> <name>Lindsay Garmon</name> </student> </course></pre> |

SAX (Simple API for XML) definition

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

SAX is an *event-driven* interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XMLparser.

SAX was developed collaboratively by members of the XML-DEV mailing list. The original version of SAX, which was specific to Java, was the first API for XML in Java to gain broad industry support.

- SAX is an event-based parsing API
 - the parser reads the XML document once
 - at each parser event it notifies the application
 - callback-style mechanism: applications must register appropriate event handlers
- SAX is lower-level than DOM – actually, DOM uses SAX – harder to use but more flexible and efficient

SAX parser for Xml

SAX parser is the most commonly used xml parser in Java after DOM, unlike DOM Sax does not loads the XML into memory before parsing it, nor it creates any type of object from XML. SAX is a better choice to parse xml's with large size. SAX uses some callback methods to parse and read the xml accordingly. It uses three callback methods listed below :

startElement() : Every time a SAX parser gets a opening tag '<', it calls startElement() and process the xml according to the code written in startElement().

endElement(): Every time a SAX parser gets a closing tag '>', it calls endElement() and process the xml according to the code written in endElement().

character(): Every time a SAX parser gets a simple character string, it calls character() method and the xml according to the code written in startElement().

The three methods explained above are responsible to parse the xml in SAX parser, developer use the methods as events and can write parsing and extraction code accordingly.

Sample Project: <http://www.beingjavaguys.com/2013/06/read-xml-file-with-sax-parser.html>

XSL

- It Started with XSL
- XSL stands for EXtensible Stylesheet Language.
- The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.

CSS = Style Sheets for HTML

HTML uses predefined tags. The meaning of, and how to display each tag is well understood. CSS is used to add styles to HTML elements.

XSL = Style Sheets for XML

XML does not use predefined tags, and therefore the meaning of each tag is not well understood. A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!

So, XSL describes how the XML elements should be displayed.

XSL - More Than a Style Sheet Language

XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

Structuring XML document using DTD

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".

Valid XML Documents

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a DTD:

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

The DOCTYPE declaration, in the example above, is a reference to an external DTD file. The content of the file is shown in the paragraph below.

XML DTD

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:

```

<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>

```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of the document is note
- !ELEMENT note defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to defines the to element to be of type "#PCDATA"
- !ELEMENT from defines the from element to be of type "#PCDATA"
- !ELEMENT heading defines the heading element to be of type "#PCDATA"
- !ELEMENT body defines the body element to be of type "#PCDATA"

Using DTD for Entity Declaration

A doctype declaration can also be used to define special characters and character strings, used in the document:

Example

```

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

<!DOCTYPE note [
<!ENTITY nbsp "&#xA0;">
<!ENTITY writer "Writer: Donald Duck.">
<!ENTITY copyright "Copyright: W3Schools.">
]>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
<footer>&writer;&nbsp;&copyright;</footer>
</note>

```

When to Use a DTD/Schema?

- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- With a DTD, you can verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

XML Schema

- An XML Schema describes the structure of an XML document, just like a DTD.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against an XML Schema is both "Well Formed" and "Valid".

XML Schema

XML Schema is an XML-based alternative to DTD:

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

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

</xs:element>
```

The Schema above is interpreted like this:

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

XML Schemas are More Powerful than DTD

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

Why Use an XML Schema?

- With XML Schema, your XML files can carry a description of its own format.
- With XML Schema, independent groups of people can agree on a standard for interchanging data.
- With XML Schema, you can verify data.

XML Schemas Support Data Types

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

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

XML Schemas use XML Syntax

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

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

XML Parsers

All modern browsers have a built-in XML parser.

An XML parser converts an XML document into an XML DOM object - which can then be manipulated with a JavaScript.

Parse an XML Document

The following code fragment parses an XML document into an XML DOM object:

```
if (window.XMLHttpRequest)
{
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
}
else
{
    // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

```
xmlhttp.open("GET","books.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
```

Parse an XML String

The following code fragment parses an XML string into an XML DOM object:

```
txt="<bookstore><book>";
txt=txt+"<title>Everyday Italian</title>";
txt=txt+"<author>Giada De Laurentiis</author>";
txt=txt+"<year>2005</year>";
txt=txt+"</book></bookstore>";

if (window.DOMParser)
{
    parser=new DOMParser();
    xmlDoc=parser.parseFromString(txt,"text/xml");
}
else // Internet Explorer
{
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async="false";
    xmlDoc.loadXML(txt);
}
```

Displaying XML with XSLT

With XSLT you can transform an XML document into HTML.

Displaying XML with XSLT

XSLT is the recommended style sheet language of XML.

XSLT (eXtensible Stylesheet Language Transformations) is far more sophisticated than CSS.

XSLT can be used to transform XML into HTML, before it is displayed by a browser:

XML CODE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<breakfast_menu>
    <food>
```

```

        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>two of our famous Belgian Waffles with plenty of real maple
syrup</description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>light Belgian waffles covered with strawberries and whipped
cream</description>
        <calories>900</calories>
    </food>
    <food>
        <name>Berry-Berry Belgian Waffles</name>
        <price>$8.95</price>
        <description>light Belgian waffles covered with an assortment of fresh berries and
whipped cream</description>
        <calories>900</calories>
    </food>
    <food>
        <name>French Toast</name>
        <price>$4.50</price>
        <description>thick slices made from our homemade sourdough bread</description>
        <calories>600</calories>
    </food>
    <food>
        <name>Homestyle Breakfast</name>
        <price>$6.95</price>
        <description>two eggs, bacon or sausage, toast, and our ever-popular hash
browns</description>
        <calories>950</calories>
    </food>
</breakfast_menu>

```

XSLT

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

```
<!-- Edited by XMLSpy -->
```

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml">
```

```
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
```

```
<xsl:for-each select="breakfast_menu/food">
```

```
<div style="background-color:teal;color:white;padding:4px">
```

```
<span style="font-weight:bold"><xsl:value-of select="name"/></span>
```

```
- <xsl:value-of select="price"/>
```

```
</div>
<div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
  <xsl:value-of select="description"/>
  <span style="font-style:italic">
    <xsl:value-of select="calories"/> (calories per serving)
  </span>
</div>
</xsl:for-each>
</body>
</html>
```

DOM

What is the Document Object Model?

The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

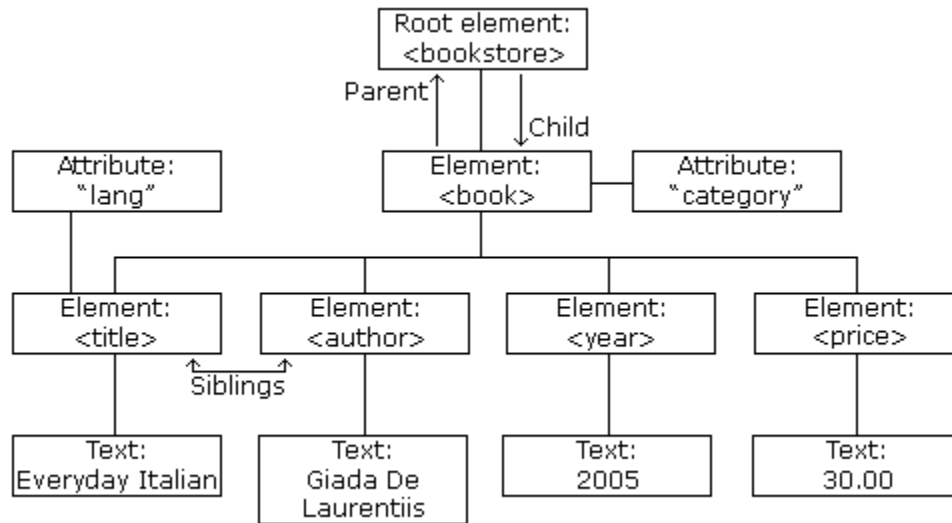
With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

The XML DOM

- The XML DOM defines a standard way for accessing and manipulating XML documents.
- The XML DOM views an XML document as a tree-structure.

All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.

XML DOM TREE EXAMPLE



The DOM is a programming API for documents. It is based on an object structure that closely resembles the structure of the documents it models. For instance, consider this table, taken from an XHTML document:

```

<table>
<tbody>
<tr>
<td>Shady Grove</td>
<td>Aeolian</td>
</tr>
<tr>
<td>Over the River, Charlie</td>
<td>Dorian</td>
</tr>
</tbody>
</table>

```

A graphical representation of the DOM of the example table, with whitespaces in element content (often abusively called "ignorable whitespace") removed, is:

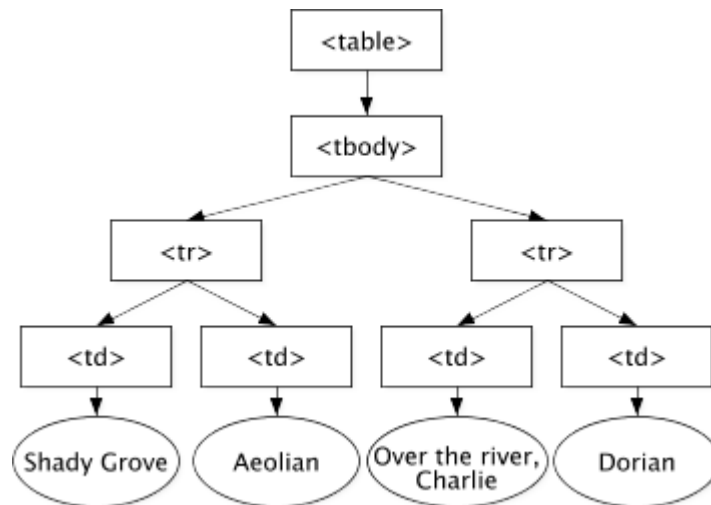


Figure: graphical representation of the DOM of the example table

An example of DOM manipulation using ECMAScript would be:

```

// access the tbody element from the table element
var myTbodyElement = myTableElement.firstChild;

// access its second tr element
// The list of children starts at 0 (and not 1).
var mySecondTrElement = myTbodyElement.childNodes[1];

// remove its first td element
mySecondTrElement.removeChild(mySecondTrElement.firstChild);

// change the text content of the remaining td element
mySecondTrElement.firstChild.firstChild.data = "Peter";

```

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, which is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one doctype nodes, one document element node, and zero or more comments or processing instructions; the document element serves as the root of the element tree for the document. However, the DOM does not specify that documents must be *implemented* as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term *structure model* to describe the tree-like representation of a document. We also use the term "tree" when referring to the arrangement of those information items which can be reached by using "tree-walking" methods; (this does not include attributes). One important property of DOM structure models is *structural isomorphism*: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, in accordance with the XML Information Set.

The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense: documents are modeled using objects, and the model

encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- The interfaces and objects used to represent and manipulate a document
- The semantics of these interfaces and objects - including both behavior and attributes
- The relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract data model, not by an object model. In an abstract data model, the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects that hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

Where the Document Object Model came from

The DOM originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the DOM Working Group was formed at W3C, it was also joined by vendors in other domains, including HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; as a result, the DOM has been influenced by SGML Groves and the HyTime standard. Some of these vendors had also developed their own object models for documents in order to provide an API for SGML/XML editors or document repositories, and these object models have also influenced the DOM.

Entities and the DOM Core

In the fundamental DOM interfaces, there are no objects representing entities. Numeric character references, and references to the pre-defined entities in HTML and XML, are replaced by the single character that makes up the entity's replacement. For example, in:

```
<p>This is a dog &amp; a cat</p>
```

the "&" will be replaced by the character "&", and the text in the P element will form a single continuous sequence of characters. Since numeric character references and pre-defined entities are not recognized as such in CDATA sections, or in the SCRIPT and STYLE elements in HTML, they are not replaced by the single character they appear to refer to. If the example above were enclosed in a CDATA section, the "&" would not be replaced by "&"; neither would the <p> be recognized as a start tag. The representation of general entities, both internal and external, are defined within the extended (XML) interfaces of Document Object Model Core.

DOM Architecture

The DOM specifications provide a set of APIs that forms the DOM API. Each DOM specification defines one or more modules and each module is associated with one feature name. For example, the DOM Core specification (this specification) defines two modules:

- The Core module, which contains the fundamental interfaces that must be implemented by all DOM conformant implementations, is associated with the feature name "Core";
- The XML module, which contains the interfaces that must be implemented by all conformant XML 1.0 (and higher) DOM implementations, is associated with the feature name "XML".

The following representation contains all DOM modules, represented using their feature names, defined along the DOM specifications:

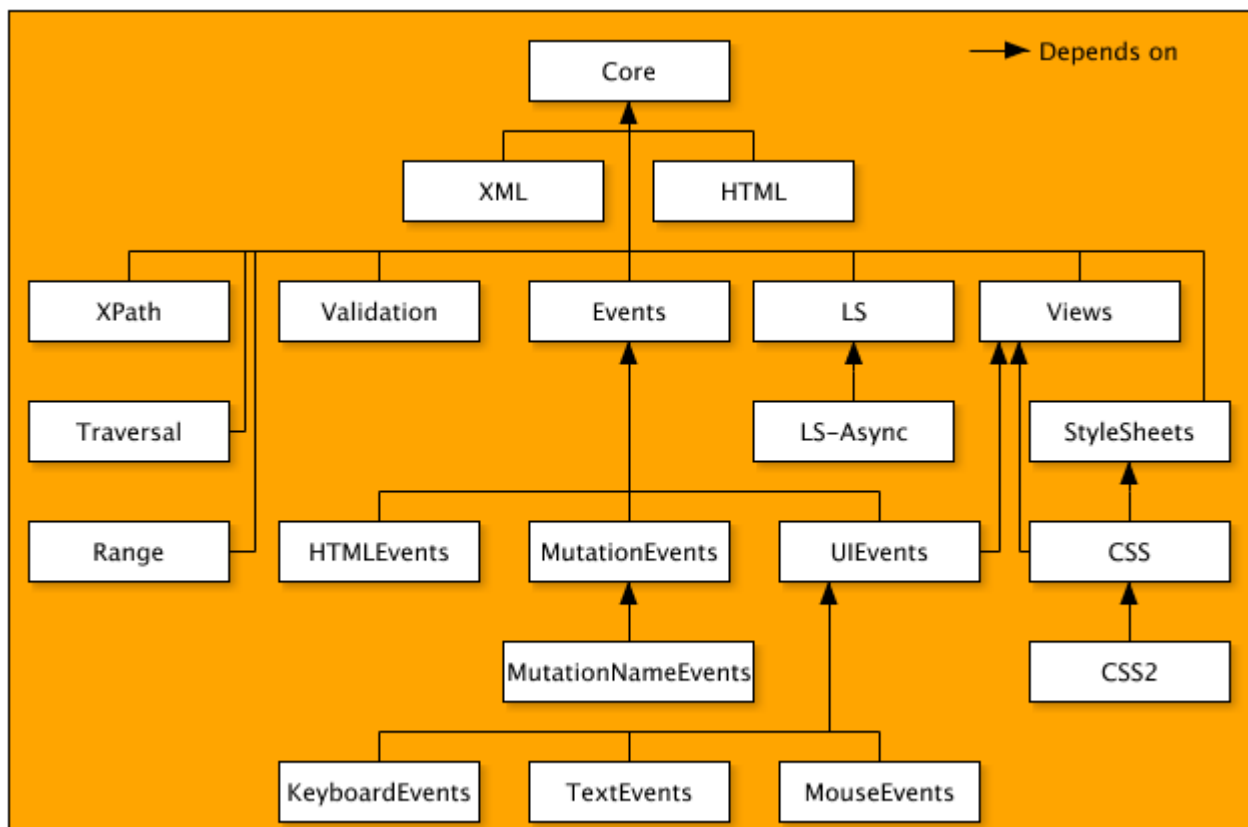


Figure: A view of the DOM Architecture

A DOM implementation can then implement one (i.e. only the Core module) or more modules depending on the host application. A Web user agent is very likely to implement the "MouseEvents" module, while a server-side application will have no use of this module and will probably not implement it.

DOM Interfaces and DOM Implementations

The DOM specifies interfaces which may be used to manage XML or HTML documents. It is important to realize that these interfaces are an abstraction - much like "abstract base classes" in C++, they are a means of specifying a way to access and manipulate an application's internal representation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in this specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies; in particular,

1. Attributes defined in the IDL do not imply concrete objects which must have specific data members - in the language bindings, they are translated to a pair of get()/set() functions, not to a data member. Read-only attributes have only a get() function in the language bindings.
2. DOM applications may provide additional interfaces and objects not found in this specification and still be considered DOM conformant.
3. Because we specify interfaces and not the actual objects that are to be created, the DOM cannot know what constructors to call for an implementation. In general, DOM users call the createX() methods on the Document class to create document structures, and DOM implementations create their own internal representations of these structures in their implementations of the createX() functions.

XForms

What Is XForms?

- XForms is the next generation of HTML forms
- XForms is richer and more flexible than HTML forms
- XForms will be the forms standard in XHTML 2.0
- XForms is platform and device independent
- XForms separates data and logic from presentation
- XForms uses XML to define form data
- XForms stores and transports data in XML documents
- XForms contains features like calculations and validations of forms
- XForms reduces or eliminates the need for scripting
- XForms is a W3C Recommendation

XForms Is The Successors Of HTML Forms

- Forms are an important part of many web applications today. An HTML form makes it possible for web applications to accept input from a user.
- Today, ten years after HTML forms became a part of the HTML standard, web users do complex transactions that are starting to exceed the limitations of standard HTML forms.
- XForms provides a richer, more secure, and device independent way of handling web input. We should expect future web solutions to demand the use of XForms-enabled browsers (All future browsers should support XForms).

XForms Separate Data From Presentation

- XForms uses XML for data definition and HTML or XHTML for data display. XForms separates the data logic of a form from its presentation. This way the XForms data can be defined independent of how the end-user will interact with the application.

XForms Uses XML To Define Form Data

- With XForms, the rules for describing and validating data are expressed in XML.
- **XForms Uses XML To Store And Transport Data**
- With XForms, the data displayed in a form are stored in an XML document, and the data submitted from the form, are transported over the internet using XML.
- The data content is coded in, and transported as Unicode bytes.
- **XForms Is Device Independent**
- Separating data from presentation makes XForms device independent, because the data model can be used for all devices. The presentation can be customized for different user interfaces, like mobile phones, handheld devices, and Braille readers for the blind.
- Since XForms is device independent and based on XML, it is also possible to add XForms elements directly into other XML applications like VoiceXML (speaking web data), WML (Wireless Markup Language), and SVG (Scalable Vector Graphics).

The XForms Framework

The purpose of an HTML form is to collect data. XForms has the same purpose.

With XForms, input data is described in two different parts:

- **The XForm model** - defines what the form is, what it should do, what data it contains
- **The XForm user interface** - defines the input fields and how they should be displayed

The XForms Model

The XForms model describes the data.

The XForms model defines a data model inside a model element:

```
<model>
  <instance>
    <person>
      <fname/>
      <lname/>
    </person>
  </instance>
  <submission id="form1" action="submit.asp" method="get"/>
</model>
```

In the example above, the XForms model uses an instance element to define the XML-template for the data to be collected, and a submission element to describe how to submit the data.

XForms Namespace

If you are missing the XForms namespace in these examples, or if you don't know what a namespace is, it will be introduced in the next chapter.

The instance Element

The instance element defines the data to be collected.

XForms is always **collecting data for an XML document**. The instance element in the XForms model defines the XML document.

In the example above the "data instance" (the XML document) the form is collecting data for looks like this:

```
<person>
  <fname/>
  <lname/>
</person>
```

After collecting the data, the XML document might look like this:

```
<person>
  <fname>John</fname>
  <lname>Smith</lname>
</person>
```

The submission Element

- The submission element describes how to submit the data.
- The submission element defines a form and how it should be submitted.

In the example above, the **id="form1"** identifies a form, the **action="submit.asp"** defines the URL to where the form should be submitted, and the **method="get"** attribute defines the method to use when submitting the form data.

The XForms User Interface

The XForms user interface defines the input fields and how they should be displayed.

The user interface elements are called controls (or input controls):

```
<input ref="fname"><label>First Name</label></input>
<input ref="lname"><label>Last Name</label></input>
<submit submission="form1"><label>Submit</label></submit>
```

In the example above the two <input> elements define two input fields. The ref="fname" and ref="lname" attributes point to the <fname> and <lname> elements in the XForms model.

The <submit> element has a submission="form1" attribute which refers to the <submission> element in the XForms model. A submit element is usually displayed as a button.

Notice the <label> elements in the example. With XForms every input control element has a required <label> element.

A Container

XForms is not designed to work alone. There is no such thing as an XForms document.

XForms has to run inside another XML document. It could run inside XHTML 1.0, and it will run inside XHTML 2.0.

If we put it all together, the document will look like this:

```
<xforms>

<model>
  <instance>
    <person>
      <fname/>
      <lname/>
    </person>
  </instance>
  <submission id="form1" action="submit.asp" method="get"/>
</model>

<input ref="fname"><label>First Name</label></input>
<input ref="lname"><label>Last Name</label></input>
<submit submission="form1"><label>Submit</label></submit>

</xforms>
```

And the page will display pretty much like this:

First Name

Last Name

The XForms Processor

An **XForms Processor** built into the browser will be responsible for submitting the XForms data to a target.

The data can be submitted as XML and could look something like this:

```
<person>
  <fname>Hege</fname>
  <lname>Refsnes</lname>
</person>
```

XForms Functions

The XForms function library includes the entire XPath 1.0 core function library.

The following table lists the additional functions within XForms:

| Function | Description |
|------------------------------------|---|
| boolean-from-string(string) | Returns true if the parameter string is "true" or "1" and false if the parameter string is "false" or "0" |
| if(boolean-test, string1, string2) | Evaluates the Boolean-test parameter and returns string1 if the test is true, and string2 if the test is false |
| avg(node-set) | Returns the average of all the nodes in the specified node-set. The value of each node is converted to a number. If the node-set is empty it returns NaN |
| min(node-set) | Returns the minimum value of all the nodes in the specified node-set. The value of each node is converted to a number. If the node-set is empty it returns NaN |
| max(node-set) | Returns the maximum value of all the nodes in the specified node-set. The value of each node is converted to a number. If the node-set is empty it returns NaN Returns: 20 |
| count-non-empty(node-set) | Returns the number of non-empty nodes in the specified node-set |
| index(string) | Returns the current index for a given repeat set |
| property(string) | Returns the property named by the string parameter <ul style="list-style-type: none">property("version") - returns the XForms version numberproperty("conformance-level") - returns the XForms conformance level ("basic" or "full") |
| now() | Returns the current system date and time in xs:dateTime |

| | |
|-------------------------------|---|
| | format |
| instance(string) | <p>An XForms Model can contain more than one instance. This function returns the root node of the specified instance data</p> <pre><xforms:instance id="orderform"> <firstName>John</firstName> </xforms:instance></pre> <p>ref="instance('orderform')/firstName"</p> <p>This example returns a node-set that consists of the firstName element node from the instance named "orderform"</p> |
| days-from-date(string) | <p>If the string parameter represents a legal xs:date or xs:dateTime, it returns the number of days between the specified date and 1970-01-01, otherwise it returns NaN</p> <p>days-from-date("2002-01-02") returns 11689 days-from-date("1969-12-29") returns -3</p> |
| seconds-from-dateTime(string) | <p>If the string parameter represents a legal xs:dateTime, it returns the number of seconds between the specified dateTime and 1970-01-01T00:00:00Z, otherwise it returns NaN</p> |
| seconds(string) | <p>If the string parameter represents a legal xs:duration, it returns the number specified in the seconds component plus 60 * the number specified in the minutes component, plus 60 * 60 * the number specified in the hours component, plus 60 * 60 * 24 * the number specified in the days component, otherwise it returns NaN</p> <p>seconds("P1Y2M") returns 0 seconds("P3DT10H30M1.5S") returns 297001.5 seconds("3") returns NaN</p> |
| months(string) | <p>If the string parameter represents a legal xs:duration, it returns the number specified in the months component plus 12 * the number specified in the years component, otherwise it returns NaN</p> <p>months("P1Y2M") returns 14 months("-P19M") returns -19</p> |

XHTML

What Is XHTML?

- XHTML stands for EXtensible HyperText Markup Language
- XHTML is almost identical to HTML
- XHTML is stricter than HTML

- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

Why XHTML?

Many pages on the internet contain "bad" HTML.

This HTML code works fine in most browsers (even if it does not follow the HTML rules):

```
<html>
<head>
  <title>This is bad HTML</title>

<body>
  <h1>Bad HTML
  <p>This is a paragraph
</body>
```

Today's market consists of different browser technologies. Some browsers run on computers, and some browsers run on mobile phones or other small devices. Smaller devices often lack the resources or power to interpret "bad" markup. XML is a markup language where documents must be marked up correctly (be "well-formed"). By combining the strengths of HTML and XML, XHTML was developed. XHTML is HTML redesigned as XML.

The Most Important Differences from HTML:

Document Structure

- XHTML DOCTYPE is **mandatory**
- The xmlns attribute in <html> is **mandatory**
- <html>, <head>, <title>, and <body> are **mandatory**

XHTML Elements

- XHTML elements must be **properly nested**
- XHTML elements must always be **closed**
- XHTML elements must be in **lowercase**
- XHTML documents must have **one root element**

XHTML Attributes

- Attribute names must be in **lower case**
- Attribute values must be **quoted**
- Attribute minimization is **forbidden**

<!DOCTYPE> Is Mandatory

An XHTML document must have an XHTML DOCTYPE declaration.

A complete list of all the XHTML Doctypes is found in our HTML Tags Reference.

The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.

This example shows an XHTML document with a minimum of required tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<title>Title of document</title>
```

```
</head>
```

```
<body>
```

```
  some content
```

```
</body>
```

```
</html>
```

XHTML Elements Must Be Properly Nested

- ✓ In HTML, some elements can be improperly nested within each other, like this:
`<i>This text is bold and italic</i>`
- ✓ In XHTML, all elements must be properly nested within each other, like this:
`<i>This text is bold and italic</i>`

XHTML Elements Must Always Be Closed

Example:

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

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

Empty Elements Must Also Be Closed

Example:

A break: `
`

A horizontal rule: `<hr />`

An image: ``

XHTML Elements Must Be In Lower Case

Example:

```
<body>
```

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

```
</body>
```

XHTML Attribute Names Must Be In Lower Case

Example:

```
<table width="100%">
```

How to Convert from HTML to XHTML

1. Add an XHTML <!DOCTYPE> to the first line of every page
2. Add an xmlns attribute to the html element of every page
3. Change all element names to lowercase
4. Close all empty elements
5. Change all attribute names to lowercase
6. Quote all attribute values

Transformations

XSLT

What is XSLT?

- XSLT stands for XSL Transformations
 - XSLT is the most important part of XSL
 - XSLT transforms an XML document into another XML document
 - XSLT uses XPath to navigate in XML documents
 - XSLT is a W3C Recommendation
-
- ✓ XSLT = XSL Transformations
 - ✓ XSLT is the most important part of XSL.
 - ✓ XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.
 - ✓ With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
 - ✓ A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree.**
 - ✓ XSLT Uses XPath
 - ✓ XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.
 - ✓ With XSLT you can transform an XML document into HTML

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT - Transformation

Example study: How to transform XML into XHTML using XSLT?

The details of this example will be explained in the next chapter.

Correct Style Sheet Declaration

The root element that declares the document to be an XSL style sheet is `<xsl:stylesheet>` or `<xsl:transform>`.

Note: `<xsl:stylesheet>` and `<xsl:transform>` are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

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

or:

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

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

The `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute `version="1.0"`.

Start with a Raw XML Document

We want to **transform** the following XML document ("cdcatalog.xml") into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
```

```

    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>

```

Viewing XML Files in IE, Chrome, Firefox, Safari, and Opera: Open the XML file (click on the link below) - The XML document will be displayed with color-coded root and child elements (except in Safari). Often, there is a plus (+) or minus sign (-) to the left of the elements that can be clicked to expand or collapse the element structure. **Tip: To view the raw XML source, right-click in XML file and select "View Source"!**

Create an XSL Style Sheet

Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template:

```

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

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

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>

```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

Link the XSL Style Sheet to the XML Document

Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

Displaying XML with XSLT

- XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
- XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- XSLT uses XPath to find information in an XML document.

XSLT Example

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
```



```

<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
<calories>900</calories>
</food>
<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped
cream</description>
<calories>900</calories>
</food>
<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>
<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
<calories>950</calories>
</food>
</breakfast_menu>

```

Use XSLT to transform XML into HTML, before it is displayed in a browser:

Example XSLT Stylesheet:

```

<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
  </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
      <xsl:value-of select="description"/>
      <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per
serving)</span>
    </p>
  </div>
</xsl:for-each>

```

```
</xsl:for-each>
</body>
</html>
```

XLINK

XLink is used to create hyperlinks in XML documents.

- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files
- XLink is a W3C Recommendation

XLink Browser Support

There is no browser support for XLink in XML documents. However, all major browsers support XLinks in SVG.

XLink Syntax

In HTML, the <a> element defines a hyperlink. However, this is not how it works in XML. In XML documents, you can use whatever element names you want - therefore it is impossible for browsers to predict what link elements will be called in XML documents.

Below is a simple example of how to use XLink to create links in an XML document:

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

<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
  <homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit
W3Schools</homepage>
  <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

- To get access to the XLink features we must declare the XLink namespace. The XLink namespace is: "http://www.w3.org/1999/xlink".
- The xlink:type and the xlink:href attributes in the <homepage> elements come from the XLink namespace.
- The xlink:type="simple" creates a simple "HTML-like" link (means "click here to go there").
- The xlink:href attribute specifies the URL to link to.

XLink Example

The following XML document contains XLink features:

```

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

<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">

  <book title="Harry Potter">
    <description
      xlink:type="simple"
      xlink:href="/images/HPotter.gif"
      xlink:show="new">
      As his fifth year at Hogwarts School of Witchcraft and
      Wizardry approaches, 15-year-old Harry Potter is.....
    </description>
  </book>

  <book title="XQuery Kick Start">
    <description
      xlink:type="simple"
      xlink:href="/images/XQuery.gif"
      xlink:show="new">
      XQuery Kick Start delivers a concise introduction
      to the XQuery standard.....
    </description>
  </book>

</bookstore>

```

Example explained:

- The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
- The xlink:type="simple" creates a simple "HTML-like" link
- The xlink:href attribute specifies the URL to link to (in this case - an image)
- The xlink:show="new" specifies that the link should open in a new window

In the example above we have demonstrated simple XLinks. XLink is getting more interesting when accessing remote locations as resources, instead of standalone pages.

If we set the value of the xlink:show attribute to "embed", the linked resource should be processed inline within the page. When you consider that this could be another XML document you could, for example, build a hierarchy of XML documents.

You can also specify WHEN the resource should appear, with the xlink:actuate attribute.

XLink Attribute Reference

| Attribute | Value | Description |
|---------------|---|---|
| xlink:actuate | onLoad onRequest other none | Defines when the linked resource is read and shown: <ul style="list-style-type: none">onLoad - the resource should be loaded and shown when the document loadsonRequest - the resource is not read or shown before the link is clicked |
| xlink:href | URL | Specifies the URL to link to |
| xlink:show | embed new replace other none | Specifies where to open the link. Default is "replace" |
| xlink:type | simple extended locator arc resource title none | Specifies the type of link |

XPATH

XPath (the XML Path language) is a language for finding information in an XML document.

What is XPath?

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is also used in XQuery, XPointer and XLink
- XPath is a W3C recommendation

XPath Path Expressions

- XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.
- Today XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

XPath is Used in XSLT

XPath is a major element in the XSLT standard. Without XPath knowledge you will not be able to create XSLT documents.

XPath Example

We will use the following XML document:

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

<bookstore>

<book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="web">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="web">
  <title lang="en">Learning XML</title>
```

```

<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>

</bookstore>

```

In the table below we have listed some XPath expressions and the result of the expressions:

| XPath Expression | Result |
|------------------------------------|--|
| /bookstore/book[1] | Selects the first book element that is the child of the bookstore element |
| /bookstore/book[last()] | Selects the last book element that is the child of the bookstore element |
| /bookstore/book[last()-1] | Selects the last but one book element that is the child of the bookstore element |
| /bookstore/book[position()<3] | Selects the first two book elements that are children of the bookstore element |
| //title[@lang] | Selects all the title elements that have an attribute named lang |
| //title[@lang='en'] | Selects all the title elements that have a "lang" attribute with a value of "en" |
| /bookstore/book[price>35.00] | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| /bookstore/book[price>35.00]/title | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

XQuery

What is XQuery?

- XQuery is *the* language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.)

XQuery is About Querying XML

- XQuery is a language for finding and extracting elements and attributes from XML documents.
- Here is an example of a question that XQuery could solve:

- "Select all CD records with a price less than \$10 from the CD collection stored in the XML document called cd_catalog.xml"

XQuery - Examples of Use

XQuery can be used to:

- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

The XML Example Document

We will use the following XML document in the examples below.

"books.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
```

```
</book>
```

```
</bookstore>
```

How to Select Nodes From "books.xml"?

Functions

XQuery uses functions to extract data from XML documents.

The doc() function is used to open the "books.xml" file:

```
doc("books.xml")
```

Path Expressions

XQuery uses path expressions to navigate through elements in an XML document.

The following path expression is used to select all the title elements in the "books.xml" file:

```
doc("books.xml")/bookstore/book/title
```

(/bookstore selects the bookstore element, /book selects all the book elements under the bookstore element, and /title selects all the title elements under each book element)

The XQuery above will extract the following:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Predicates

XQuery uses predicates to limit the extracted data from XML documents.

The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30:

```
doc("books.xml")/bookstore/book[price<30]
```

The XQuery above will extract the following:

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


How to Select Nodes From "books.xml" With FLWOR

Look at the following path expression:

```
doc("books.xml")/bookstore/book[price>30]/title
```

The expression above will select all the title elements under the book elements that are under the bookstore element that have a price element with a value that is higher than 30.

The following FLWOR expression will select exactly the same as the path expression above:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

The result will be:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

With FLWOR you can sort the result:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

FLWOR is an acronym for "For, Let, Where, Order by, Return".

- The **for** clause selects all book elements under the bookstore element into a variable called \$x.
- The **where** clause selects only book elements with a price element with a value greater than 30.
- The **order by** clause defines the sort-order. Will be sort by the title element.
- The **return** clause specifies what should be returned. Here it returns the title elements.
- The result of the XQuery expression above will be:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

References:

1. Heather Williamson, "The Complete Reference XML", Tata Mc Graw Hill, 2001.
2. <http://www.w3schools.com>
3. <http://www.tutorialspoint.com/>

Unit-III

Client-Side Scripting

Introduction

Client-side scripting generally refers to the class of computer programs on the web that are executed *client-side*, by the user's web browser, instead of *server-side*. This type of computer programming is an important part of the Dynamic HTML (DHTML) concept, enabling web pages to be scripted; that is, to have different and changing content depending on user input, environmental conditions (such as the time of day), or other variables.

Client-side scripts are often embedded within an HTML or XHTML document (hence known as an "embedded script"), but they may also be contained in a separate file, to which the document (or documents) that use it to make reference (hence known as an "external script"). Upon request, the necessary files are sent to the user's computer by the web server (or servers) on which they reside. The user's web browser executes the script, then displays the document, including any visible output from the script. Client-side scripts may also contain instructions for the browser to follow in response to certain user actions, (e.g., clicking a button). Often, these instructions can be followed without further communication with the server. Balancing the execution between client and server scripts is used to minimize the communication load, server load and/or response time

Client-side scripting is not inherently unsafe. Users, though, are encouraged to always keep their web browsers up-to-date to avoid exposing their computer and data to new vulnerabilities. Some users disable script execution because of fear of exploits or to pursue faster browser experience, thus limiting the usefulness of this scripting technique.

The latest group of web browsers and web pages tend to employ a heavy amount of client-side scripting, accounting for an improved user interface in which the user does not experience the unfriendly "refreshing" of the web page, but instead sees perhaps an animated GIF file indicating that the request occurred and the page will be updated shortly. Ajax is an important addition to the JavaScript language, allowing web developers to communicate with the web server asynchronously in the background without requiring a completely new version of the page to be requested and rendered. This leads to a much improved user experience in general.

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 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 contain 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>
```

Advantages

- Javascript is executed on the client side
- Javascript is a relatively easy language
- Javascript is relatively fast to the end user

- Extended functionality to web pages

Disadvantages

- Security Issues
- Javascript rendering varies

Data Types

JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

```
var length = 10;           // Number
var lastName = "John";    // String
var cars = ["Saab", "Volvo", "BMW"]; // Array
var x = {firstName:"John", lastName:"Doe"}; // Object
```

Booleans:

```
var x = true;
var y = false;
```

Arrays: Example

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

Objects: Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

typeof Operator: Example

```
typeof "John"      // Returns string
typeof 3.14        // Returns number
typeof false       // Returns boolean
typeof [1,2,3,4]   // Returns object
typeof {name:'John', age:34} // Returns object
```

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.

Operators

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 ";

Control 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 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>

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>
```

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>
```

JavaScript Events

Events are actions that can be detected by JavaScript.

Event Association

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

| 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. |
| onDblClick | 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. |

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.

Objects and Arrays

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 than 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 contain 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 are 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 "fooooo".

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 "fooooo", but not "fod" or "foooooo".

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 "foooooo", but not "fd".

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 |

| | |
|-----------|--|
| | document.write() or document.writeln() |
| write() | Writes HTML expressions or JavaScript code to a document |
| writeln() | Same as write(), 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 a 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.shift("limit","lines")
```

The array, words, will be:

limit, lines,finish, complete, In, Out

Windows and frames

JavaScript is an *object-oriented* (or, as some would argue, *object-based*) language.

An object is a set of variables, functions, etc., that are in some way related. They are grouped together and given a name

Objects may have:

Properties

A variable (numeric, string or Boolean) associated with an object. Most properties can be changed by the user.

Example: the title of a document

Methods

Functions associated with an object. Can be called by the user.

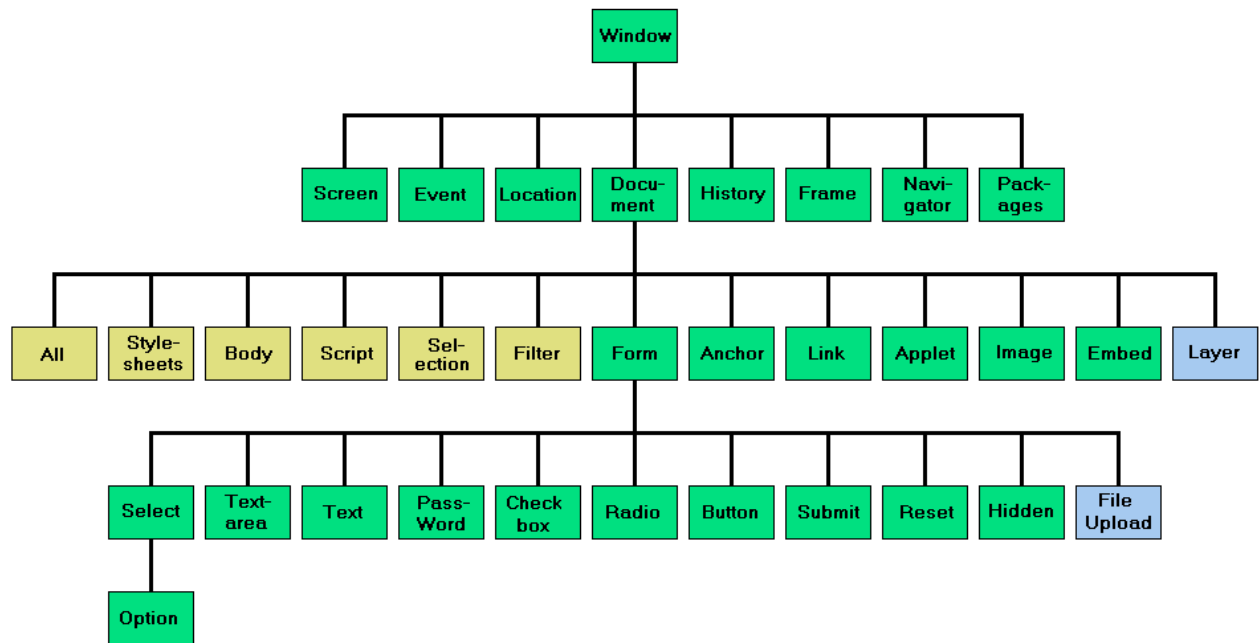
Example: the `alert()` method

Events

Notification that a particular event has occurred. Can be used by the programmer to trigger responses.

Example: the `onClick()` event.

The objects are arranged into a hierarchy as shown below:



The Document Object Model. Objects shown in green are common to both Netscape Navigator and Internet Explorer; objects shown in yellow are found only in Internet Explorer while objects shown in blue are found only in Netscape Navigator.

The hierarchy of objects is known as the Document Object Model (DOM).

Window Object

Window is the fundamental object in the browser. It represents the browser window in which the document appears

Its **properties** include:

status

The contents of the status bar (at the bottom of the browser window). For example:

```
window.status = "Hi, there!";
```

will display the string "Hi, there!" on the status bar.

[Click here to see this line of code in operation.](#)

location

The location and URL of the document currently loaded into the window (as displayed in the location bar). For example:

```
alert(window.location);
```

will display an alert containing the location and URL of this document.

[Click here to see this line of code in operation.](#)

`length`

The number of frames (if any) into which the current window is divided. For example:

```
alert(window.length);
```

will display an alert indicating the number of frames in the current window.

See under `parent` (below) for an example.

`parent`

The parent window, if the current window is a sub-window in a frameset. For example:

```
var parentWindow = window.parent;
alert(parentWindow.length);
```

will place a string representing the parent window into the variable `parentWindow`, then use it to report the number of frames (if any) in the parent window.

`top`

The top-level window, of which all other windows are sub-windows. For example:

```
var topWindow = window.top;
alert(topWindow.length);
```

will place a string representing the top-level window into the variable `topWindow`, then use it to report the number of frames (if any) in the top-level window.

`top` behaves in a very similar way to `parent`. Where there are only two levels of windows, `top` and `parent` will both indicate the same window. However, if there are more than two levels of windows, `parent` will indicate the parent of the current window, which may vary depending upon which window the code is in. However, `top` will always indicate the very top-level window.

Window **methods** include:

`alert()` Displays an 'alert' dialog box, containing text entered by the page designer, and an 'OK' button. For example:

```
alert("Hi, there!");
```

will display a dialog box containing the message "Hi, there!".

[Click here to see this line of code in operation.](#)

`confirm()` Displays a 'confirm' dialog box, containing text entered by the user, an 'OK' button, and a 'Cancel' button. Returns `true` or `false`. For example:

```
var response = confirm("Delete File?");  
alert(response);
```

will display a dialog box containing the message "Delete File?" along with an 'OK' button and a 'Cancel' button. If the user clicks on 'OK' the variable `response` will contain the Boolean value `true`, and this will appear in the 'alert' dialog-box; If the user clicks on 'Cancel' the variable `response` will contain the Boolean value `false` and this will appear in the 'alert' dialog-box.

`prompt()` Displays a message, a box into which the user can type text, an 'OK' button, and a 'Cancel' button. Returns a text string. The syntax is:

```
prompt(message_string, default_response_string)
```

For example:

```
var fileName = prompt("Select File", "file.txt");  
alert(fileName);
```

will display a dialog box containing the message "Select File" along with an 'OK' button, a 'Cancel' button, and an area into which the user can type. This area will contain the string "file.txt", but this can be overwritten with a new name. If the user clicks on 'OK' the variable `fileName` will contain the string "file.txt" or whatever the user entered in its place, and this will be reported

using an alert dialog box.

`open()`

Opens a new browser window and loads either an existing page or a new document into it. The syntax is:

```
open(URL_string, name_string, parameter_string)
```

For example:

```
var parameters = "height=100,width=200";  
newWindow = open("05_JS4nw.html", "newDocument",  
parameters);
```

will open a new window 100 pixels high by 200 pixels wide. An HTML document called '05_JS4nw.html' will be loaded into this window.

Note that the variable `newWindow` is not preceded by the word `var`. This is because `newWindow` is a **global variable** which was declared at the start of the script. The reason for this is explained below.

[Click here to see this code in operation.](#)

`close()`

Closes a window. If no window is specified, closes the current window. The syntax is:

```
window_name.close()
```

For example:

```
newWindow.close()
```

will close the new window opened by the previous example.

Note that the window name (i.e., `newWindow`) must be declared as a global variable if we want to open the window using one function and close it using another function. If it had been declared as a **local variable**, it would be lost from the computer's memory as soon as the first function ended, and we would not then

be able to use it to close the window.

[Click here to see this code in operation.](#)

Window **events** include:

`onLoad()` Message sent each time a document is loaded into a window. Can be used to trigger actions (e.g., calling a function). Usually placed within the `<body>` tag, for example:

```
<body onLoad="displayWelcome()">
```

would cause the function `displayWelcome()` to execute automatically every time the document is loaded or refreshed.

`onUnload()` Message sent each time a document is closed or replaced with another document. Can be used to trigger actions (e.g., calling a function). Usually placed within the `<body>` tag, for example:

```
<body onUnload="displayFarewell()">
```

would cause the function `displayFarewell()` to execute automatically every time the document is closed or refreshed.

Frames- an array of frames in the window

The window history can be used to go back and forth on the list of visited pages. For example,

```
history.back(); // reloads the previous page
```

```
history.forward(); // reloads the next page
```

```
history.go(-3); // goes back three pages
```

For a frames page, a page with a frameset element, each frame is also represented by its own window object. These objects are kept on the frames array. Thus give the window objects for the frames using their name or id attributes.

```
frames["top"]
```

```
frames["left"]
```

```
frames["right"]
```

Forms

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 fl()
{
    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=fl() />
</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 f1()
{
    var age=myForm.t1.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=f1 () />
</form>
</body>
</html>

```

Output of the program

Enter Age:

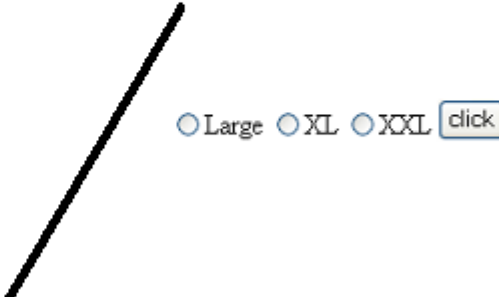
The above program will accept the input only in any one of the following valid form
25 5 101

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>

```



Validate form selection

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.

AJAX

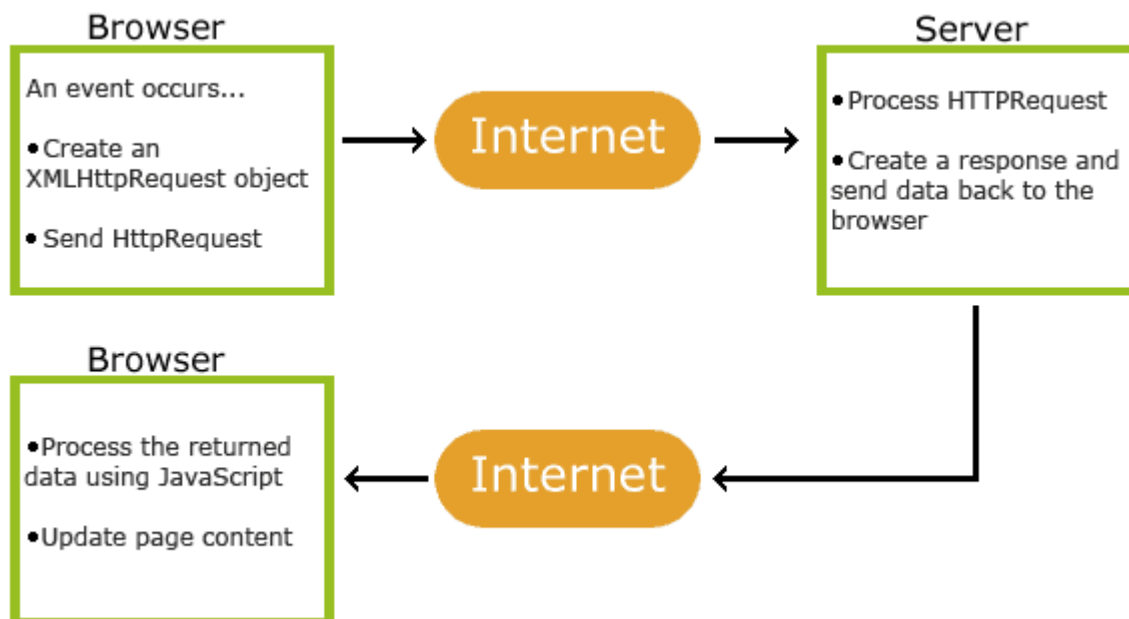
What is AJAX?

AJAX = Asynchronous JavaScript and XML.

- AJAX is a technique for creating fast and dynamic web pages.

- 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.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook.

How AJAX Works



AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to retrieve data from a web server)
- JavaScript/DOM (to display/use the data)

AJAX Example

HTML page

```

<!DOCTYPE html>
<html>
<body>

<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>
  
```

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

The HTML page contains a <div> section and a <button>.
The <div> section is used to display information from a server.
The <button> calls a function (if it is clicked).
The function requests data from a web server and displays it:

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      document.getElementById("demo").innerHTML = xhttp.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

XMLHttpRequest (XHR)

- The keystone of AJAX is the XMLHttpRequest object.
- All modern browsers support the XMLHttpRequest object.
- The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create Object

Create an XMLHttpRequest Object

All modern browsers (Chrome, IE7+, Firefox, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:
variable = new XMLHttpRequest();

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX Object:
variable = new ActiveXObject("Microsoft.XMLHTTP");

To handle all browsers, including IE5 and IE6, check if the browser supports the XMLHttpRequest object. If it does, create an XMLHttpRequest object, if not, create an ActiveXObject:

Example

```
var xhttp;  
if (window.XMLHttpRequest) {  
    xhttp = new XMLHttpRequest();  
} else {  
    // code for IE6, IE5  
    xhttp = new ActiveXObject("Microsoft.XMLHTTP");  
}
```

The XMLHttpRequest object is used to exchange data with a server.

Request

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);  
xhttp.send();
```

| Method | Description |
|---|--|
| | Specifies the type of request |
| open(<i>method</i> , <i>url</i> , <i>async</i>) | <i>method</i> : the type of request: GET or POST <i>url</i> : the server (file) location <i>async</i> : true (asynchronous) or false (synchronous) |
| send() | Sends the request to the server (used for GET) |
| send(<i>string</i>) | Sends the request to the server (used for POST) |

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

GET Requests

A simple GET request:

Example

```
xhttp.open("GET", "demo_get.asp", true);  
xhttp.send();
```

In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

Example

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random(), true);  
xhttp.send();
```

POST Requests

A simple POST request:

Example

```
xhttp.open("POST", "demo_post.asp", true);  
xhttp.send();
```

To POST data like an HTML form, add an HTTP header with `setRequestHeader()`. Specify the data you want to send in the `send()` method:

Example

```
xhttp.open("POST", "ajax_test.asp", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("fname=Henry&lname=Ford");
```

Response

Server Response

To get the response from a server, use the `responseText` or `responseXML` property of the `XMLHttpRequest` object.

| Property | Description |
|---------------------------|-----------------------------------|
| <code>responseText</code> | get the response data as a string |
| <code>responseXML</code> | get the response data as XML data |

The `responseText` Property

If the response from the server is not XML, use the `responseText` property.

The `responseText` property returns the response as a string, and you can use it accordingly:

Example

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The responseXML Property

If the response from the server is XML, and you want to parse it as an XML object, use the `responseXML` property:

Example

Request the file `cd_catalog.xml` and parse the response:

```
xmlDoc = xhttp.responseXML;  
txt = "";  
x = xmlDoc.getElementsByTagName("ARTIST");  
for (i = 0; i < x.length; i++) {  
    txt += x[i].childNodes[0].nodeValue + "<br>";  
}  
document.getElementById("demo").innerHTML = txt;
```

Ready State

The `readyState` property holds the status of the `XMLHttpRequest`.

Holds the status of the `XMLHttpRequest`. Changes from 0 to 4:

- 0: request not initialized
- 1: server connection established
- 2: request received
- 3: processing request
- 4: request finished and response is ready

jQuery

- jQuery is a fast, small, and feature-rich JavaScript library.
- It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.
- With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Features:

jQuery includes the following features:

- DOM element selections using the multi-browser open source selector engine *Sizzle*, a spin-off of the jQuery project.
- DOM manipulation based on CSS selectors that uses elements' names and attributes, such as id and class, as criteria to select nodes in the DOM
- Events
- Effects and animations
- AJAX
- Deferred and Promise objects to control asynchronous processing
- Utilities, such as feature detection
- Compatibility methods that are natively available in modern browsers, but need fall backs for older ones, such as `inArray()` and `each()`
- Multi-browser support

Example:

```
<html>
<head>
<title>The jQuery Example</title>
<script type="text/javascript" src="/jquery/jquery-2.1.3.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
document.write("Hello, World!");
});
</script>
</head>
<body>
<h1>Hello</h1>
</body>
</html>
```

References:

1. Jeff frantzen and Sobotka, “Java script “, Tata McGraw hill.
2. <http://www.w3schools.com>
3. <http://www.jscripters.com>
4. <http://www.tutorialspoint.com>

Unit-IV

Server-Side Scripting

Introduction

Server-side scripting is a technique used in web development which involves employing scripts on a web server which produce a response customized for each user's (client's) request to the website. The alternative is for the web server itself to deliver a static web page. Scripts can be written in any of a number of server-side scripting languages. Server-side scripting is distinguished from client-side scripting where embedded scripts, such as JavaScript, are run client-side in a web browser, but both techniques are often used together.

Server-side scripting is often used to provide a customized interface for the user. These scripts may assemble client characteristics for use in customizing the response based on those characteristics, the user's requirements, access rights, etc. Server-side scripting also enables the website owner to hide the source code that generates the interface, whereas with client-side scripting, the user has access to all the code received by the client. A down-side to the use of server-side scripting is that the client needs to make further requests over the network to the server in order to show new information to the user via the web browser. These requests can slow down the experience for the user, place more load on the server, and prevent use of the application when the user is disconnected from the server.

When the server serves data in a commonly used manner, for example according to the HTTP or FTP protocols, users may have their choice of a number of client programs (most modern web browsers can request and receive data using both of those protocols). In the case of more specialized applications, programmers may write their own server, client, and communications protocol, that can only be used with one another.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be considered client-side operations.

Differences between Client-side and Server-side Scripting

Client-side Environment

The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web server to the users computer over the internet and run directly in the browser.

The scripting language needs to be **enabled** on the client computer. Sometimes if a user is conscious of **security risks** they may switch the scripting facility off. When this is the case a message usually pops up to alert the user when script is attempting to run.

Server-side Environment

The **server-side environment** that runs a scripting language is a web server. A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server.

This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

ASP

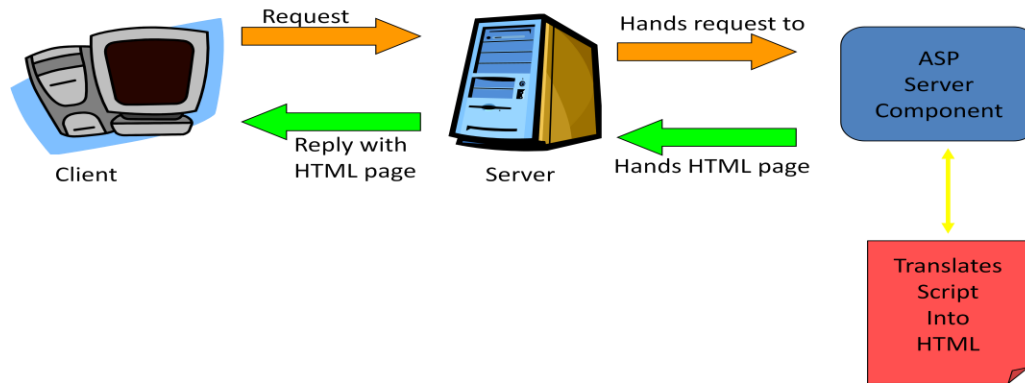
What is ASP?

- ASP stands for Active Server Pages
- ASP is a Microsoft Technology
- ASP is a program that runs inside Internet Information Services (IIS)
- IIS comes as a free component with Windows 2000
- IIS is also a part of the Windows NT 4.0 Option Pack
- The Option Pack can be downloaded from Microsoft
- Personal Web Server (PWS) is a smaller - but fully functional - version of IIS
- PWS can be found on your Windows 95/98 CD
- Active Server Page was developed by Microsoft and it is a popular technology for developing dynamic web sites.
- Must run on an active server pages server
 - IIS, PWS, ...
- The latest version is ASP.NET

What is an ASP File?

- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML, XML, and Scripts
- Scripts in an ASP file are executed on the Server
- An ASP file has the file extension ".asp"

How to load an ASP page?



List of Objects

- Request Object
- Response Object
- Server Object
- Session Object
- Application Object

Request Object

Definition: The request object gets the information from client browser while HTTP request

Syntax: Request.[collection|property|method] (variable).

- Collections
 - ClientCertificate
 - To get certification fields from client side
 - Cookies
 - To get the values from cookies during HTTP request
 - Form
 - To get the values from form elements during HTTP request
 - QueryString
 - To get the values of variables during HTTP request
 - Server Variables
 - To get the values of predetermined environment variables
- Properties
 - Read-only
 - Size of the content being sent by client in terms of bytes.
- Methods
 - Binary Read
 - It reads the information sent by client while POST request
- In case collection/property/method is not mentioned, directly variable name is used to fetch the data, ASP engine follows the below mentioned hierarchy to search the variable
 1. QueryString
 2. Form

3. Cookies
4. ClientCertificate
5. ServerVariables

Response Object

Definition: This object is used to send the information to client from server.

Syntax:

Response.[collection|method|property]

- Collections
 - Cookies
 - The response object supports only cookies as collection to set cookie values.
- Properties
 - Buffer
 - to indicate whether page output is buffered
 - CacheControl
 - Determine whether proxy servers are able to cache the output generated by ASP
 - Charset
 - Appends the name of the character set to content-type header
 - ContentType
 - Specifies the HTTP content type for the response
 - ExpiresAbsolute
 - specifies the date and time when page should expire on client side
 - IsClientConnected
 - Indicates whether the client is connected to server or not
 - Pics
 - Platform for Internet Content Selection
 - It is used to rate the content in a website
 - This property appends a value to PICS label response header
 - Status
 - Value of status line returned by the server
 - Flush
 - Sends buffered output immediately
 - Redirect
 - Sends a redirect message to browser
 - Write
 - Writes the content to HTTP output

Server Object

Definition: it provides access to methods and properties on the server

Syntax: Server.property|method

- Properties
 - ScriptTimeout
 - Amount of time that a script can run before it times out
- Methods
 - Execute
 - Used to run another asp file within asp file
 - CreateObject
 - Creates an instance of server component
 - HTMLEncode
 - Applies HTML encoding to string
 - MapPath
 - Maps the virtual path into physical path
 - URLEncode
 - Applies URL encoding rules to string

Session Object

Definition: it is used to store the temporary info for user. Data available in sessions remain available within the website while visiting one page to another page. Sessions are automatically created whenever user visits the website

Syntax:

Session.[collection|method|property]

- Collections
 - Contents
 - Contains the data which are added by script
 - StaticObjects
 - Contains the object created with <object> tag and given session scope
- Properties
 - CodePage
 - Codepage is used for symbol mapping
 - LCID
 - The locale identifier
 - SessionID
 - Returns the session identification of user
 - Timeout
 - Timeout period for session
- Methods
 - Abandon
 - This method destroys a session object and releases its resources

Application Object

Definition: This object can store info regarding the application which remains constant throughout the whole time server is running

Syntax: Application.method

- Collections

- Content
 - Contains all the items that have been added by script
- StaticObjects
 - Contains all the data for objects created by <OBJECT> tag
- Lock
 - This method stops other clients from modifying application data
- Unlock
 - This method allows the client for modification of application data

File Access

File Object

The File object is used to return information about a specified file.

- Attributes- Sets or returns the attributes of a specified file.
- DateCreated -Returns the date and time when a specified file was created
- DateLastAccessed- Returns the date and time when a specified file was last accessed
- DateLastModified Returns the date and time when a specified file was last modified
- Drive Returns the drive letter of the drive where a specified file or folder resides
- Name Sets or returns the name of a specified file
- Path Returns the path for a specified file
- ShortName Returns the short name of a specified file (the 8.3 naming convention)
- ShortPath Returns the short path of a specified file
- Size Returns the size, in bytes, of a specified file
- Type Returns the type of a specified file
- ParentFolder Returns the folder object for the parent of the specified file

Methods

- Copy Copies a specified file from one location to another
- Delete Deletes a specified file
- Move Moves a specified file from one location to another
- OpenAsTextStream Opens a specified file and returns a TextStream object to access the file

ASP AdRotator Component

- The ASP AdRotator component creates an AdRotator object that displays a different image each time a user enters or refreshes a page.
- A text file includes information about the images.

Syntax

```
<%
    set
    adrotator=server.createobject("MSWC.AdRotator")
    adrotator.GetAdvertisement("textfile.txt")
%>
```

ASP AdRotator Properties

Border :

- Specifies the size of the borders around the advertisement

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.Border="2"
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

Clickable:

- Specifies whether the advertisement is a hyperlink

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.Clickable=false
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

Target:

- Frame Name of the frame to display the advertisement

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.TargetFrame="target='_blank'"
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

ASP AdRotator Methods**GetAdvertisement**

- Returns HTML that displays the advertisement in the page
- ```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

**ASP Content Linking Component**

- The ASP Content Linking component is used to create a quick and easy navigation system!
- The Content Linking component returns a Nextlink object that is used to hold a list of Web pages to be navigated.

**Syntax**

```
<%
Set nl=Server.CreateObject("MSWC.NextLink")
%>
```

**Example**

- First we create a text file - "links.txt":
- asp\_intro.asp ASP Intro
- asp\_syntax.asp ASP Syntax

asp\_variables.asp ASP Variables  
asp\_procedures.asp ASP Procedures

#### **GetListCount**

- Returns the number of items listed in the Content Linking List file
- ```
<%  
dim nl,c  
Set nl=Server.CreateObject("MSWC.NextLink")  
c=nl.GetListCount("links.txt")  
Response.Write("There are ")  
Response.Write(c)  
Response.Write(" items in the list")  
%>Output:  
There are 4 items in the list
```

GetListIndex

- Returns the index number of the current item in the Content Linking List file. The index number of the first item is 1. 0 is returned if the current page is not in the Content Linking List file
- ```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetListIndex("links.txt")
Response.Write("Item number ")
Response.Write(c)
%>
Output:
Item number 3
```

#### **GetNextDescription**

- Returns the text description of the next item listed in the Content Linking List file. If the current page is not found in the list file it returns the text description of the last page on the list
- ```
<%  
dim nl,c  
Set nl=Server.CreateObject("MSWC.NextLink")  
c=nl.GetNextDescription("links.txt")  
Response.Write("Next ")  
Response.Write("description is: ")  
Response.Write(c)  
%>  
Next description is: ASP Variables
```

GetNextURL

- Returns the URL of the next item listed in the Content Linking List file. If the current page is not found in the list file it returns the URL of the last page on the list

- <%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNextURL("links.txt")
Response.Write("Next ")
Response.Write("URL is: ")
Response.Write(c)
%>
- Next URL is: asp_variables.asp

GetNextURL

- Returns the URL of the next item listed in the Content Linking List file. If the current page is not found in the list file it returns the URL of the last page on the list
- <%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNextURL("links.txt")
Response.Write("Next ")
Response.Write("URL is: ")
Response.Write(c)
%>
- Next URL is: asp_variables.asp

GetNthURL

- Returns the URL of the Nth page listed in the Content Linking List file
- <%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNthURL("links.txt",3)
Response.Write("Third ")
Response.Write("URL is: ")
Response.Write(c)
%>
- Third URL is: asp_variables.asp

GetPrevious

- Description Returns the text description of the previous item listed in the Content Linking List file. If the current page is not found in the list file it returns the text description of the first page on the list
- <%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetPreviousDescription("links.txt")
Response.Write("Previous ")
Response.Write("description is: ")
Response.Write(c)
%>
Previous description is: ASP Variables

GetPreviousURL

- Returns the URL of the previous item listed in the Content Linking List file. If the current page is not found in the list file it returns the URL of the first page on the list
- ```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetPreviousURL("links.txt")
Response.Write("Previous ")
Response.Write("URL is: ")
Response.Write(c)
%>
```
- Previous URL is: asp\_variables.asp

### **ASP Content Rotator Component**

- The ASP Content Rotator component creates a ContentRotator object that displays a different content string each time a visitor enters or refreshes a page.
- A text file, called the Content Schedule File, includes the information about the content strings.
- The content strings can contain HTML tags so you can display any type of content that HTML can represent: text, images, colors, or hyperlinks.
- Syntax
- ```
<%
Set cr=Server.CreateObject("MSWC.ContentRotator")
```

Example:

- The following example displays a different content each time a visitor views the Web page.
- First, create a text file named "textads.txt" and place it in a subfolder called "text".
- "textads.txt":

```
%% #3 <h2>This is a great day!!</h2>

%% #3 

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

ASP Content Rotator Component's Methods

ChooseContent

- Gets and displays a content string
- ```
<%
dim cr
Set cr=Server.CreateObject("MSWC.ContentRotator")
response.write(cr.ChooseContent("text/textads.txt"))
%>
```

#### **Output:**



### GetAllContent

- Retrieves and displays all of the content strings in the text file
- `<%  
dim cr  
Set cr=Server.CreateObject("MSWC.ContentRotator")  
response.write(cr.GetAllContent("text/textads.txt"))  
%>`
- Output:
- This is a great day!!



### ASP Browser Capabilities Component

- It component creates a BrowserType object that find outs the type, capabilities and version number of a visitor's browser.
- The BrowserType object compares the information in the header with information in a file on the server called "Browscap.ini".
- if there is no match for the browser type and version number in the Browscap.ini file, it will set every property to "UNKNOWN".
- If there is a match between the browser type and version number in the header and the information in the "Browscap.ini" file, list the properties of the matching browser.

#### Syntax:

- `<%  
Set MyBrow=Server.CreateObject("MSWC.BrowserType")  
%>`

```
<html><body>
<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>
<table border="0" width="100%"><tr>
<th>Client OS</th><th><%=MyBrow.platform%></th>
</tr><tr><td>Web Browser</td> <td><%=MyBrow.browser%></td> </tr><tr>
<td>Browser version</td><td><%=MyBrow.version%>
</td></tr><tr>
<td>Frame support?</td><td><%=MyBrow.frames%></td>
</tr><tr>
<td>Table support?</td><td><%=MyBrow.tables%></td>
</tr><tr>
<td>Sound support?</td><td><%=MyBrow.backgroundsounds%>
</td></tr><tr>
<td>Cookies support?</td><td><%=MyBrow.cookies%>
</td></tr><tr>
<td>VBScript support?</td><td><%=MyBrow.vbscript%>
</td></tr></tr>
```

```

<td>JavaScript support?</td><td><%=MyBrow.javascript%>
</td> </tr>
</table>
</body>
</html>

```

Output:

- Client OS                      WinNT
- Web Browser                IE
- Browser version            5.0
- Frame support?            True
- Table support?            True
- Sound support?            True
- Cookies support?        True
- VBScript support?        True
- JavaScript support? True

## **Accessing database**

### **Server Side Scripting Model:**

This model is characterized by the inclusion of labels or specific marks inside the documents (HTML, XML or WML) located the server, inside those, orders required for the server to execute are included, after that an HTML (or XML/WML) answer is generated which is understandable for the client's browser.

The general process that is followed:

- The browser specifies the URL from the document to be executed.
- A request is sent to Internet or the host of the Web server.
- The Web server executes the script (included inside the labels in the document or in an independent file).
- The script communicates with an external application and it recovers data. - The script returns the output to the Web server.
- The Web server sends the output from the script to the client's browser.

The main advantages of this type of solutions include the independence in connection with the client's browser; the use of SQL language to get the interaction with databases, and the easiness and short time of test.

### **Disadvantages:**

Its main disadvantages reside in the sacrifice in performance because the scripts are interpreted (they require that the server carries out syntactic revision in each execution) and the necessity to modify the code of the scripts when there are changes carried out in the interface. This model is characteristic of solutions as ASP, Cold Fusion and PHP, and other solutions.

ASP: This is the solution of Microsoft which can be implemented in servers with Windows NT with Internet Information Server of Microsoft, and inclusive for Windows 95/98 with Personal Web Server installed.

The main characteristics are:

- The access to databases is carried out by means of ODBC and objects ADO (Active Data Objects) that are placed in the scripts.
- Simple of to create and to update, then it does not need too much experience in programming. - It allows to manage sessions and therefore the storage of variables can remain during different connections.
- It allows the use of different script languages (inclusive combined inside the same file) like: VBScript (subset of Visual Basic), JavaScript, PerlScript.

### **Disadvantages:**

- They have been broadly proven in servers IIS of Microsoft, and only until recently they have been begun to develop products for other platforms (as iASP for Linux).
- When being interpreted, performance is sacrificed, because syntactic revision is made in each execution (This restrictive one can be overcome using Web classes, which allow to create compiled objects, but it will be required the acquisition of a programming language that supports this technology additionally) Next a ASP script is presented, it visualizes data of a database Access, that contains a chart friends with two fields: telephone and name, for which it has been created a DSN (Data Source Name) previously in the control panel denominated aliasamigos.

## **PHP**

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

### **Common uses of PHP**

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.

- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

## Characteristics of PHP

Five important characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

## "Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this :

```
<html>
 <head>
 <title>Hello World</title>
 </head>
 <body>
 <?php echo "Hello, World!";?>
 </body>
</html>
```

Output:

Hello, World!

```
<!DOCTYPE HTML>
```

```
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>
```

PHP has a total of eight data types to construct our variables

- **Integers** – are whole numbers, without a decimal point, like 4195.
- **Doubles** – are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** – have only two possible values either true or false.
- **NULL** – is a special type that only has one value: NULL.
- **Strings** – are sequences of characters, like 'PHP supports string operations.'
- **Arrays** – are named and indexed collections of other values.
- **Objects** – are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** – are special variables that hold references to resources external to PHP (such as database connections).

### **PHP - GET & POST Methods**

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

### **PHP Forms Example**

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
 if (empty($_POST["name"])) {
 $nameErr = "Name is required";
 } else {
 $name = test_input($_POST["name"]);
 // check if name only contains letters and whitespace
 if (!preg_match("/^[a-zA-Z]*$/",$name)) {
 $nameErr = "Only letters and white space allowed";
 }
 }
}

if (empty($_POST["email"])) {
 $emailErr = "Email is required";
```

```

 } else {
 $email = test_input($_POST["email"]);
 // check if e-mail address is well-formed
 if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
 $emailErr = "Invalid email format";
 }
 }

 if (empty($_POST["website"])) {
 $website = "";
 } else {
 $website = test_input($_POST["website"]);
 // check if URL address syntax is valid (this regular expression also allows dashes in the
 URL)
 if (!preg_match("/^b(?:(:https?|ftp):\\\/\\\/www\\.)?[-a-z0-9+&@#\\/%?~_!:,;]*[-a-z0-9+&@#\\/%?~_]/i",$website)) {
 $websiteErr = "Invalid URL";
 }
 }

 if (empty($_POST["comment"])) {
 $comment = "";
 } else {
 $comment = test_input($_POST["comment"]);
 }

 if (empty($_POST["gender"])) {
 $genderErr = "Gender is required";
 } else {
 $gender = test_input($_POST["gender"]);
 }
}

function test_input($data) {
 $data = trim($data);
 $data = stripslashes($data);
 $data = htmlspecialchars($data);
 return $data;
}
?>

```

<h2>PHP Form Validation Example</h2>

<p><span class="error">\* required field.</span></p>

<form method="post" action="<?php echo htmlspecialchars(\$\_SERVER["PHP\_SELF"]);?>">

    Name: <input type="text" name="name" value="<?php echo \$name;?>">

    <span class="error">\* <?php echo \$nameErr;?></span>

```


E-mail: <input type="text" name="email" value="<?php echo $email;?>">
* <?php echo $emailErr;?>

Website: <input type="text" name="website" value="<?php echo $website;?>">
<?php echo $websiteErr;?>

Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>

Gender:
<input type="radio" name="gender" <?php if (isset($gender) && $gender=="female") echo
"checked";?> value="female">Female
<input type="radio" name="gender" <?php if (isset($gender) && $gender=="male") echo
"checked";?> value="male">Male
* <?php echo $genderErr;?>

<input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "
";
echo $email;
echo "
";
echo $website;
echo "
";
echo $comment;
echo "
";
echo $gender;
?>

</body>
</html>

```

## PHP Form Validation Example

\* required field.

Name: \*

E-mail: \*

Website:



Comment: 

Gender: ☐ Female ☐ Male \*

**Submit**

### **Sending a mail**

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail\_from which defines your own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = webmaster@tutorialspoint.com
```

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail\_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =

; For win32 only
sendmail_from =

; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

## **Sending plain text email**

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the message and the actual message additionally there are other two optional parameters.

mail( to, subject, message, headers, parameters );

Here is the description for each parameters.

| <b>Sr.No</b> | <b>Parameter &amp; Description</b>                                                                                            |
|--------------|-------------------------------------------------------------------------------------------------------------------------------|
|              | <b>to</b>                                                                                                                     |
| 1            | Required. Specifies the receiver / receivers of the email                                                                     |
|              | <b>subject</b>                                                                                                                |
| 2            | Required. Specifies the subject of the email. This parameter cannot contain any newline characters                            |
|              | <b>message</b>                                                                                                                |
| 3            | Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters |
|              | <b>headers</b>                                                                                                                |
| 4            | Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
|              | <b>parameters</b>                                                                                                             |
| 5            | Optional. Specifies an additional parameter to the send mail program                                                          |

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

## **Sending HTML email**

When you send a text message using PHP then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message you can specify a Mime version, content type and character set to send an HTML email.

## Example

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>

<head>
 <title>Sending HTML email using PHP</title>
</head>

<body>

 <?php
 $to = "xyz@somedomain.com";
 $subject = "This is subject";

 $message = "This is HTML message.";
 $message .= "<h1>This is headline.</h1>";

 $header = "From:abc@somedomain.com \r\n";
 $header = "Cc:afgh@somedomain.com \r\n";
 $header .= "MIME-Version: 1.0\r\n";
 $header .= "Content-type: text/html\r\n";

 $retval = mail ($to,$subject,$message,$header);

 if($retval == true) {
 echo "Message sent successfully...";
 }else {
 echo "Message could not be sent...";
 }
 ?>

</body>
</html>
```

## References:

- Scot Johnson, Keith Ballinger, Davis Chapman, "Using Active Server Pages", Prentice Hall of India.
- <http://www.w3schools.com/>
- <http://www.tutorialspoint.com>

## **Unit-V**

### **Web applications**

A web application is any application that uses a web browser as a client. The application can be as simple as a message board or a guest sign-in book on a website, or as complex as a word processor or a spreadsheet.

#### **What is a Client?**

The 'client' is used in client-server environment to refer to the program the person uses to run the application. A client-server environment is one in which multiple computers share information such as entering information into a database.

The 'client' is the application used to enter the information, and the 'server' is the application used to store the information.

#### **What are the Benefits of a Web Application?**

A web application relieves the developer of the responsibility of building a client for a specific type of computer or a specific operating system. Since the client runs in a web browser, the user could be using an IBM-compatible or a Mac. They can be running Windows XP or Windows Vista. They can even be using Internet Explorer or Firefox, though some applications require a specific web browser.

Web applications commonly use a combination of server-side script (ASP, PHP, etc) and client-side script (HTML, Javascript, etc.) to develop the application.

#### **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 Application Security**

**Web application security** is a branch of Information Security that deals specifically with security of websites, web applications and web services. At a high level, Web application security draws on the principles of application security but applies them specifically to Internet and Web systems.

#### **Security threats**

With the emergence of Web 2.0, increased information sharing through social networking and increasing business adoption of the **Web** as a means of doing business and delivering service,

websites are often attacked directly. Hackers either seek to compromise the corporate network or the end-users accessing the website by subjecting them to drive-by downloading.

### **Attacks**

- cross-site scripting (XSS)
- SQL injection attacks
- Phishing

### **Security Standards**

**OWASP** : The **Open Web Application Security Project (OWASP)** is an online community which creates freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security

### **Risks Associated with Web Applications**

An applications are often riddled with vulnerabilities that are used by attackers to gain access to either the web server or the database server. From there any number of things can happen. They can:

- Deface a web site
- Insert spam links directing visitors to another site
- Insert malicious code that installs itself onto a visitor's computer
- Insert malicious code that steals session IDs (cookies)
- Steal visitor information and browsing habits
- Steal account information
- Steal information stored in the database
- Access restricted content

### **Preventing Web Application Attacks**

- With dotDefender web application firewall you can avoid many different threats to web applications because dotDefender inspects your HTTP traffic and checks their packets against rules such as to allow or deny protocols, ports, or IP addresses to stop web applications from being exploited.
- Architected as plug & play software, dotDefender provides optimal out-of-the-box protection against DoS threats, cross-site scripting, SQL Injection attacks, path traversal and many other web attack techniques.

### **Database access and mapping**

### **Accessing Databases from Web Applications**

Data that is shared between web components and is persistent between invocations of a web application is usually maintained in a database. Web applications use the JDBC API to access relational databases. In the JDBC API, databases are accessed via DataSource objects. A DataSource has a set of properties that identify and describe the real world data source that it

represents. These properties include information such as the location of the database server, the name of the database, the network protocol to use to communicate with the server, and so on.

Web applications access a data source using a connection, and a `DataSource` object can be thought of as a factory for connections to the particular data source that the `DataSource` instance represents. In a basic `DataSource` implementation, a call to the `getConnection` method returns a connection object that is a physical connection to the data source. In the Application Server, a data source is referred to as a JDBC resource.

If a `DataSource` object is registered with a JNDI naming service, an application can use the JNDI API to access that `DataSource` object, which can then be used to connect to the data source it represents.

This section describes how to

- Populate the database with bookstore data
- Create a data source in the Application Server
- Specify a web application's resource reference
- Map the resource reference to the data source defined in the Application Server

## Populating the Example Database

Note: Application Server 8.2 includes a copy of the open source Derby database server. Application Server 8.0/ 8.1 includes the PointBase database server. If you are using Application Server 8.0/8.1, either follow the instructions in the J2EE Tutorial at <http://java.sun.com/j2ee/1.4/docs/tutorial-update6/doc/index.html> that works with Application Server 8.0/8.1 or upgrade to Application Server 8.2 (see <http://java.sun.com/j2ee/1.4/download.html#appserv> to download).

To populate the database for the Duke's Bookstore examples, follow these steps:

1. Start the Application Server, if it has not been started.
2. In a terminal window, go to `<INSTALL>/j2eetutorial14/examples/web/bookstore/`.
3. Run `asant create-db_common`. This task starts the database, if it has not been started, and executes the SQL commands contained in the `books.sql` file.
4. At the end of the processing, you should see the following output:

```
...
[sql] Executing file:
 <j2eetutorial.home>\examples\web\bookstore\books.sql
[sql] 8 of 8 SQL statements executed successfully
```

## Creating a Data Source in the Application Server

Data sources in the Application Server implement connection pooling. To define the Duke's Bookstore data source, you use the installed Derby connection pool named `DerbyPool`.

You create the data source using the Application Server Admin Console, following this procedure:

1. Expand the JDBC node.
2. Select the JDBC Resources node.
3. Click the New... button.
4. Type jdbc/BookDB in the JNDI Name field.
5. Choose DerbyPool for the Pool Name.
6. Click OK.

## Specifying a Web Application's Resource Reference

To access a database from a web application, you must declare a resource reference in the application's web application deployment descriptor. The resource reference specifies a JNDI name, the type of the data resource, and the kind of authentication used when the resource is accessed. To specify a resource reference for a Duke's Bookstore example using deploytool, follow these steps:

1. Select the WAR.
2. Select the Resource Ref's tab.
3. Click Add.
4. Type jdbc/BookDB in the Coded Name field.
5. Accept the default type javax.sql.DataSource.
6. Accept the default authorization Container.
7. Accept the default Sharable selected.

To create the connection to the database, the data access object `database.BookDBAO` looks up the JNDI name of the bookstore data source object:

```
public BookDBAO () throws Exception {
 try {
 Context initCtx = new InitialContext();
 Context envCtx = (Context)
 initCtx.lookup("java:comp/env");
 DataSource ds = (DataSource) envCtx.lookup("jdbc/BookDB");
 con = ds.getConnection();
 System.out.println("Created connection to database.");
 } catch (Exception ex) {
 System.out.println("Couldn't create connection." +
 ex.getMessage());
 throw new
 Exception("Couldn't open connection to database: "
 + ex.getMessage());
 }
}
```

## Mapping the Resource Reference to a Data Source

Both the web application resource reference and the data source defined in the Application Server have JNDI names. To connect the resource reference to the data source, you must map the

JNDI name of the former to the latter. This mapping is stored in the web application runtime deployment descriptor. To create this mapping using `deploytool`, follow these steps:

1. Select localhost:4848 in the Servers list to retrieve the data sources defined in the Application Server.
2. Select the WAR in the Web WARs list.
3. Select the Resource Ref's tab.
4. Select the Resource Reference Name, `jdbc/BookDB`, defined in the previous section.
5. In the Sun-specific Settings frame, select `jdbc/BookDB` from the JNDI Name drop-down list.

## **Web template system**

A **web template system** uses a template processor to combine web templates to form finished web pages, possibly using some data source to customize the pages or present a large amount of content on similar-looking pages. It is a web publishing tool present in content management systems, web application frameworks, and HTML editors.

Web templates can be used like the template of a form letter to either generate a large number of "static" (unchanging) web pages in advance, or to produce "dynamic" web pages on demand.

### **Applications**

Web templates can be used by any individual or organization to set up their website. Once a template is purchased or downloaded, the user will replace all generic information included in the web template with their own personal, organizational or product information.

### **Examples:**

- Display personal information or daily activities as in a blog.
- Sell products on-line.
- Display information about a company or organization.
- Display family history.
- Display a gallery of photos.
- Place music files such as MP3 files on-line for play through a web browser.
- Place videos on-line for public viewing.
- To set up a private login area on-line.

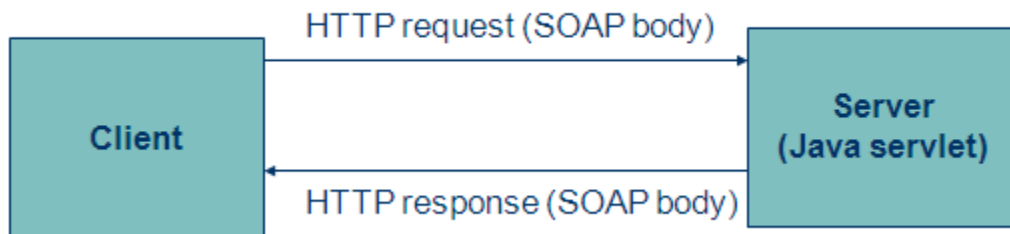
## **Web Services**

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

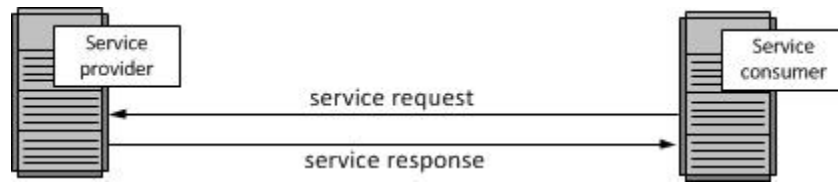
#### Standard web services technologies:

- SOAP
- WDSL
- UDDI

### **Service-Oriented Architecture**

- ♦ A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.
- ♦ Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification. For more on DCOM and CORBA.
- ♦ The technology of Web Services is the most likely connection technology of service-oriented architectures.
- ♦ The following figure illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both

the service consumer and service provider. A service provider can also be a service consumer.



## Elements of SOA

- Application frontend
- Service
- Service repository
- Service bus
- Contract
- Implementation
- Interface
- Business logic
- Data

## **SOAP (Simple Object Access Protocol)**

The basic web services platform is XML plus HTTP.

- SOAP is a communication protocol.
- SOAP is for communication between applications.
- SOAP is a format for sending messages.
- SOAP is designed to communicate via internet.
- SOAP is platform independent.
- SOAP is language independent.
- SOAP is based on XML.
- SOAP is simple and extensible.
- SOAP allows you to get around firewalls.
- SOAP will be developed as a W3C standard.

## **The SOAP Message:**

A one-way message, a request from a client, or a response from a server is referred to as SOAP message.

Every SOAP message has a mandatory Envelope element, an optional Header element, and a mandatory Body element. Each of the elements has an associated set of rules.

## **Envelope:**

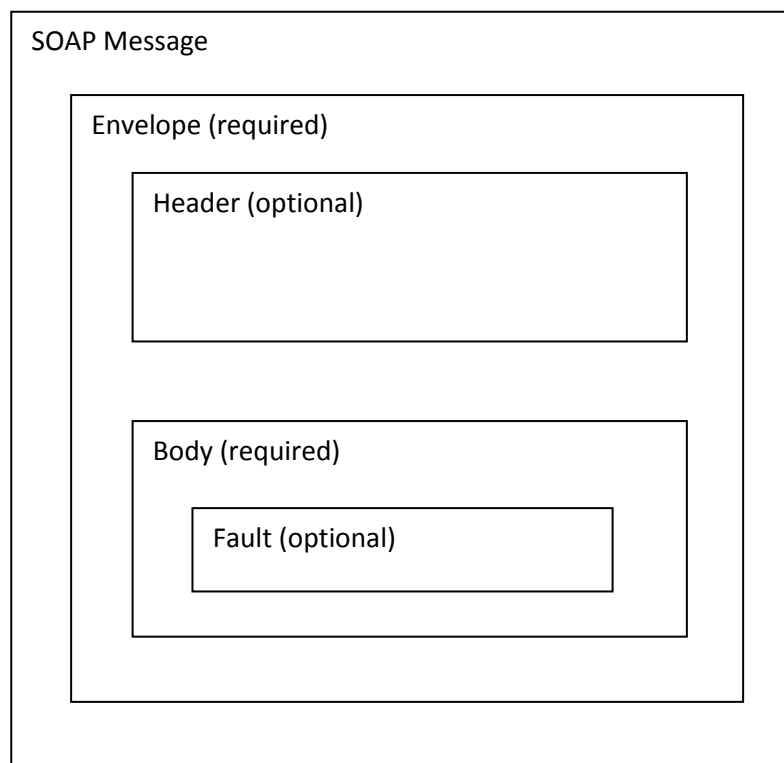
Every SOAP message has a root Envelope element. SOAP does not define a versioning. Rather, SOAP uses XML namespaces to differentiate versions.

The SOAP1.1 namespace URL is <http://schemas.xmlsoap.org/soap/envelope> , whereas SOAP1.2 namespace URL is <http://www.w3.org/2001/09/soap-envelope>.

If the envelope is in any other namespace, it is considered a versioning error.

<SOAP-ENV: Envelope xmlns: SOAP-ENV=”  
<http://schemas.xmlsoap.org/soap/envelope/> “>

Main elements of the XML SOAP Message.



## **Header:**

The optional Header element offers a flexible framework for specifying additional application level requirements.

For example, the Header element can be used to specify a digital signature for password-protected services. The Header framework provides an open mechanism for authentication, transaction management and payment authentications.

### **Header Attributes:**

- Actor – The SOAP protocol defines a message path as a list of SOAP service nodes.

Each of these intermediate nodes can perform some processing and forward the message to the next node in the chain.

By setting the actor attribute, the client can specify the recipient of the SOAP header.

- Must Understand – Indicates whether a Header element is optional or mandatory. If set to true, the recipient must understand process the Header attribute.

```
<SOAP-ENV: Header>
```

```
<ns: PaymentAccount xmlns: ns="urn:..."
```

```
SOAP-ENV: mustUnderstand="true">
```

```
</ns: PaymentAccount>
```

```
</SOAP-ENV: Header>
```

### **Body:**

The Body element is mandatory for all SOAP messages.

### **Fault:**

In the event of an error, the Body element will include a Fault element.

The Fault sub elements include the faultCode, faultString, faultActor, and detail elements.

#### **faultCode:**

A text code used to indicate a class of errors.

#### **faultString:**

A human-readable explanation of the error.

#### **faultActor:**

A text string indicating who caused the fault. This is usable if the SOAP message travels through several nodes in the SOAP message, and the clients to know which node caused the error.

**Detail:**

An element used to carry application-specific error message.

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

```
<SOAP-ENV: Envelope
```

```
 xmlns: SOAP-ENV=" http://schemas.xmlsoap.org/soap/envelope/ "
```

```
 xmlns: xsi=" http://www.w3.org/1999/xmlschema-instance "
```

```
 xmlns: xsd=" http://www.w3.org/1999/xmlschema ">
```

```
<SOAP-ENV: Body>
```

```
 <SOAP-ENV: Fault>
```

```
 <faultCode xsi: type="xsd: string">
```

```
 SOAP-ENV: client
```

```
 </faultCode>
```

```
 <faultString xsi: type="xsd: string">
```

```
 Failed to locate method (validate credit card)
```

```
 </faultString>
```

```
 </SOAP-ENV: Fault>
```

```
</SOAP-ENV: Body>
```

```
</SOAP-ENV: Envelope>
```

**SOAP Encoding:**

- SOAP includes a built-in set of rules for encoding data types.
- This enables the SOAP messages to indicate specific data types such as integers, floats, doubles or arrays.
- Most of the time, the encoding rules are implemented directly by the SOAP.

- SOAP data types are divided into two broad categories scalar types and compound types.
- Scalar types contain exactly one value, such as a last name, price, or product description.
- Compound types contain multiple values such as a purchase order.
- The encoding style for a SOAP message is set via the SOAP-ENV: encoding style attribute.
- To use SOAP 1.1 encoding, use the value <http://schemas.xmlsoap.org/soap/encoding/>
- To use SOAP 1.2 encoding, use the value <http://www.w3.org/2001/09/soap-encoding>

### **SOAP Request:**

The client request must include the name of the method to invoke and any parameter. Here is a simple client request sent to Xmethod.

First, the request includes a single mandatory Envelope element which in turn includes mandatory Body elements.

Second, a total of four XML namespaces are defined. Namespaces are used to disambiguate XML elements and attributes. The body element encapsulates the main "pay load" of the SOAP message. The only element is getTemp, which is tied to the XMethods namespace and corresponds to the remote method name.

Each parameter to the method appears as a sub element. Here we have a single zip code element which is assigned to the XML schema xsd: string data type and set to 10016.

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

```
<SOAP-ENV: Envelope
```

```
 xmlns:SOAP-env="http://schemas.xmlsoap.org/soap/envelope/
```

```
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
```

```
<SOAP-ENV: Body>
```

```
 <ns1: getTemp xmlns: ns1="urn: xmethod-Temperature"
```

```
 SOAP-ENV: encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/ ">
```

```
 <zipcode xsi: type=xsd: string>10016</zipcode>
```

```
</ns1: getTemp>
```

</SOAP-ENV: Body>

</SOAP-ENV: Envelope>

### **SOAP Response:**

Here is the SOAP response from XMethods:

<? xml version="1.0">

<SOAP-ENV: Envelope

xmlns : SOAP-ENV=" <http://schemas.xmlsoap.org/soap/envelope/> “

xmlns: xsi=" <http://www.w3.org/2001/xmlschema-instance> “

xmlns: xsd=" <http://www.w3.org/2001/xmlschema> “>

<SOAP-ENV: Body>

<ns1: getTempResponse

xmlns: ns1="urn: xmethods-Temperature”

SOAP-ENV:encodingStyle="http://schema.xmlSOAP.org/SOAP/encoding/">

<return xsi: type="xsd: float">71.0</return>

</ns1: getTempResponse>

</SOAP-ENV: body>

</SOAP-ENV: envelope>

Just like the request, the response includes envelope and body elements and same four xml namespaces. The body element includes a single getTempResponse element, corresponding to our initial request. The response element includes a single return element, indicating an xsd: float.

### **SOAP TRANSPORT:**

- SOAP does not care what transport protocol is used to exchange the message.
- This makes SOAP extremely flexible in how and where it is used.
- To exchange SOAP message through HTTP, FTP, raw TCP, SMTP, POP3.

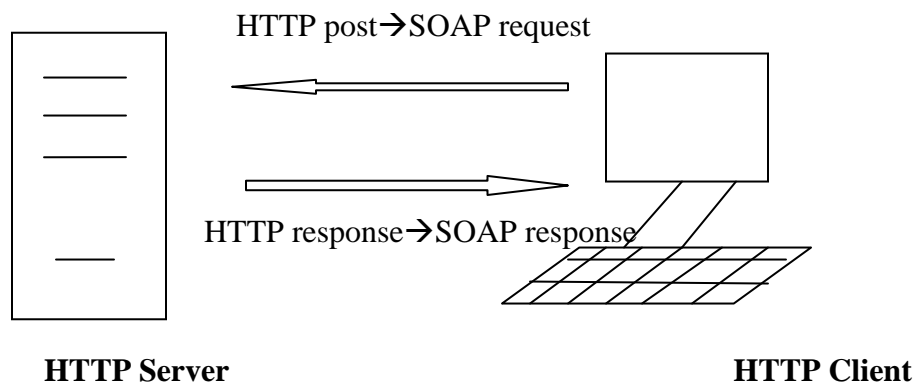
### **SOAP Over HTTP:**

(Currently HTTP is the most popular option for Service because of the frequency on the internal, transport)

- HTTP is by far the most common transport used to exchange SOAP messages.
- HTTP is simple, stable and widely deployed.
- Most Firewall allows HTTP Traffic.

SOAP-over-HTTP is a natural match with SOAP's RPC (request-response) conventions because HTTP is a request-response based protocol.

The SOAP request message is posted to the HTTP server with the HTTP request, and the Server returns the SOAP response message in the HTTP response.



HTTP request containing a SOAP message

```
POST/target HTTP/1.0
Content_Type: text/xml
Content_Length:
SOAPAction: "urn: StockQuote#GetQuote"
<SOAP_Env
```

HTTP response containing a SOAP message

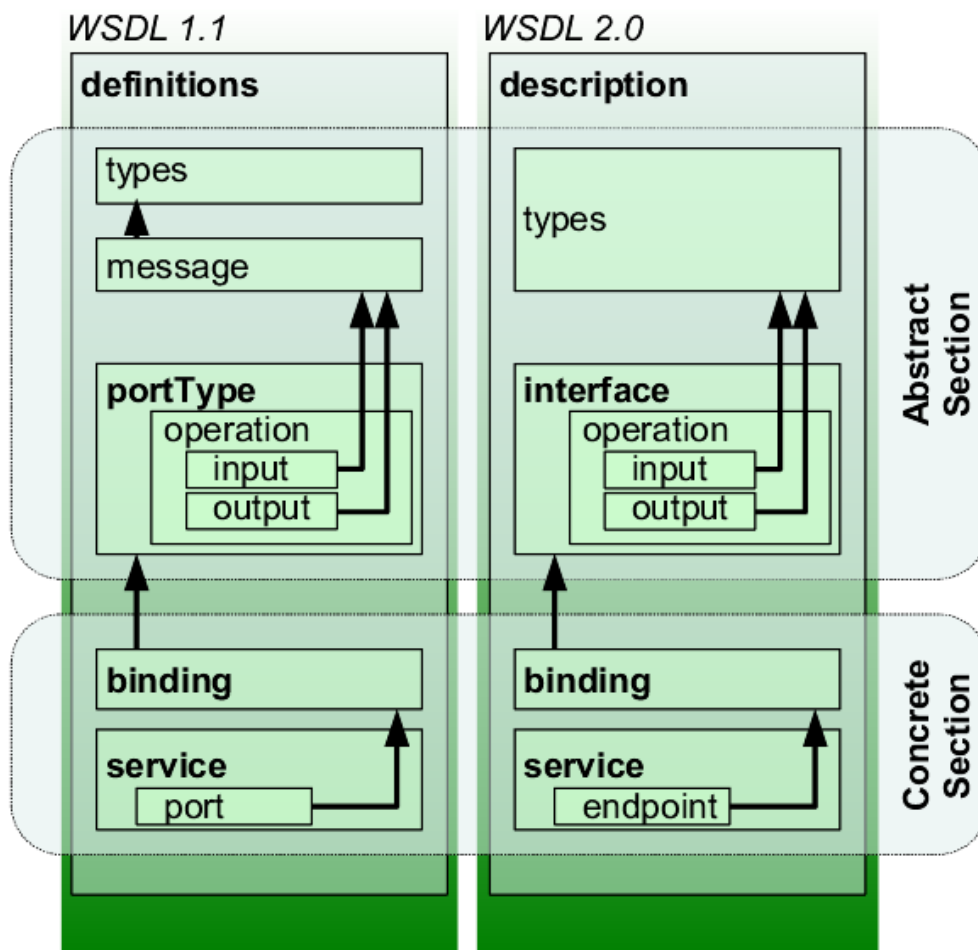
```
HTTP/1.1 200 ok
Content_Type: text/xml
Content_Length:
```



The SOAPAction HTTP header is defined by the SOAP specification and indicates the intent of the SOAP HTTP request.

Its value is completely arbitrary but is intended to tell the HTTP server what the SOAP message wants to do before the HTTP server decodes the XML.

## WSDL



### Basic wsdl example:

#### Helloservice.wsdl

#### Definitions:

The definitions element specifies that this document is the helloservice. It also specifies many namespaces that will be used through-out the document. The use of namespaces is important for differentiating elements. It enables the document preference multiple external

specification, including the wsdl specification, the SOAP specification and the xml schema specification.

The definitions element also specifies a target namespace attribute that is a conversion of xml schema that enables the wsdl document to refer itself.

We specified a target namespace of “<http://www.ecerami.com/wsdl/helloservice.wsdl>”

```
<definitions name="helloservice">
```

```
 target namespace="http://www.ecerami.com/wsdl/helloservice.wsdl"
```

```
 xmlns="http://schemas.xmlsoap.org/wsdl/"
```

```
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
```

```
 xmlns="http://www.ecerami.com/wsdl/helloservice.wsdl"
```

```
 "xmlns:xsd="http://www.w3.org/2001/XMLSchema" ">
```

### **Methods:**

Two messages elements are defined. The first represents a request message, sayhellorequest and the second represents a response message sayhelloresponse.

```
<message name="sayhellorequest">
```

```
 <part name="firstname" type="xsd: string"/>
```

```
</message>
```

```
<message name="sayhelloresponse">
```

```
 <part name="greeting" type="xsd: string"/>
```

```
</message>
```

Each of these messages contains a single part element. For the request, the part specifies the function parameters; we specify a single firstname parameter. For the response, the part specifies the function return value. The part element's type attribute specifies an xml schema data type.

If the function expects multiple arguments or return multiple values, we can specify multiple part element.

### **Port type:**

The port type element defines a single operation called sayhello. The operation itself consists of a single input message (sayhellorequest) and a single output message (sayhelloresponse)

```

<port type name="hello_port type">
 <operation name="sayhello">
 <input message="tns: sayhellorequest"/>
 <input message="tns: sayhelloresponse"/>
 </operation>
</port type>

```

### **Binding:**

The binding element provides specific details on how a port type operation will actually be transmitted over the wire. Bindings can be made available via multiple transports including http get, http post, or SOAP. The binding elements itself specifies the name and type attributes.

```

<binding name="hello_binding" type="tns: hello_port type">

```

The type attribute reference the port type defined in the document.

### **SOAP binding:**

Wsd1 1.1 includes built-in extensions for SOAP 1.1. This enables you to specify SOAP-specific details, including SOAP header, SOAP encoding styles, and the SOAP action http header. The SOAP extension elements include,

### **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. This means that the body of the SOAP request will indicate a wrapper xml element indicating the function name. A style value of document specifies an XML document call format.

The transport attribute indicates the transport of the SOAP messages. The value <http://schemas.xmlsoap.org/soap/http> indicates the SOAP http transport.

### **SOAP: Operation**

This element indicates the binding of a specific operation to a specific SOAP implementation. The SOAP Action attribute specifies that the SOAP Action HTTP header be used for identifying the service.

## **SOAP: Body**

This element enables you to specify the details of the input and output messages.

```
<binding name="Hello_Binding" type="tns: Hello_PortType">
 <SOAP: binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
 <operation name="sayHello">
 <SOAP: operation SOAPAction="sayHello"/>
 <input>
 <SOAP: body encodingstyle="http://schemas.xmlsoap.org/soap/encoding"
 namespace="URN: examples: helloservice" use="encoded">
 </input>
 <output>
 <SOAP: body encodingstyle="http://schemas.xmlsoap.org/soap/encoding"
 namespace="URN: examples: helloservice" use="encoded">
 </output>
 </operation>
 </binding>
```

## **Service**

The service element specifies the location of the service. Because this is a SOAP service, we use the SOAP: address element and specify the localhost address for the apache SOAP rpc router servlet `http://localhost:8080/soap/servlet/rpcrouter`

```
<service name="Hello_Service">
 <documentation>WSDL file for Hello Service</documentation>
 <port binding="tns: Hello_Binding" name="Hello_port">
 <SOAP: address location="http://localhost:8080/soap/servlet/rpc router"/>
 </port>
</service>
```

## **UDDI**

- UDDI is a directory service where business can register and search for web services.
- UDDI stands for Universal Description Discovery and Integration.
- UDDI is a directory for storing information about webservice.
- UDDI was originally created by Microsoft, IBM and Ariba, represents a technical specification for publishing and finding business and webservices.
- UDDI consists of two parts. First, UDDI is a technical specification for building a distributed directory of business and web service.
- Data is stored within a specific XML format. The UDDI specification includes API details for searching existing data and publishing new data. Second, the UDDI business

registry is a fully operational implementation of UDDI specification. The data captured within UDDI divided into three main categories.

### **WHITE PAGES:**

White pages provide very limited information such as name, telephone and contact address.

### **YELLOW PAGES:**

Yellow pages provide a categorization based on business and service type. For example, the data may include industry, product

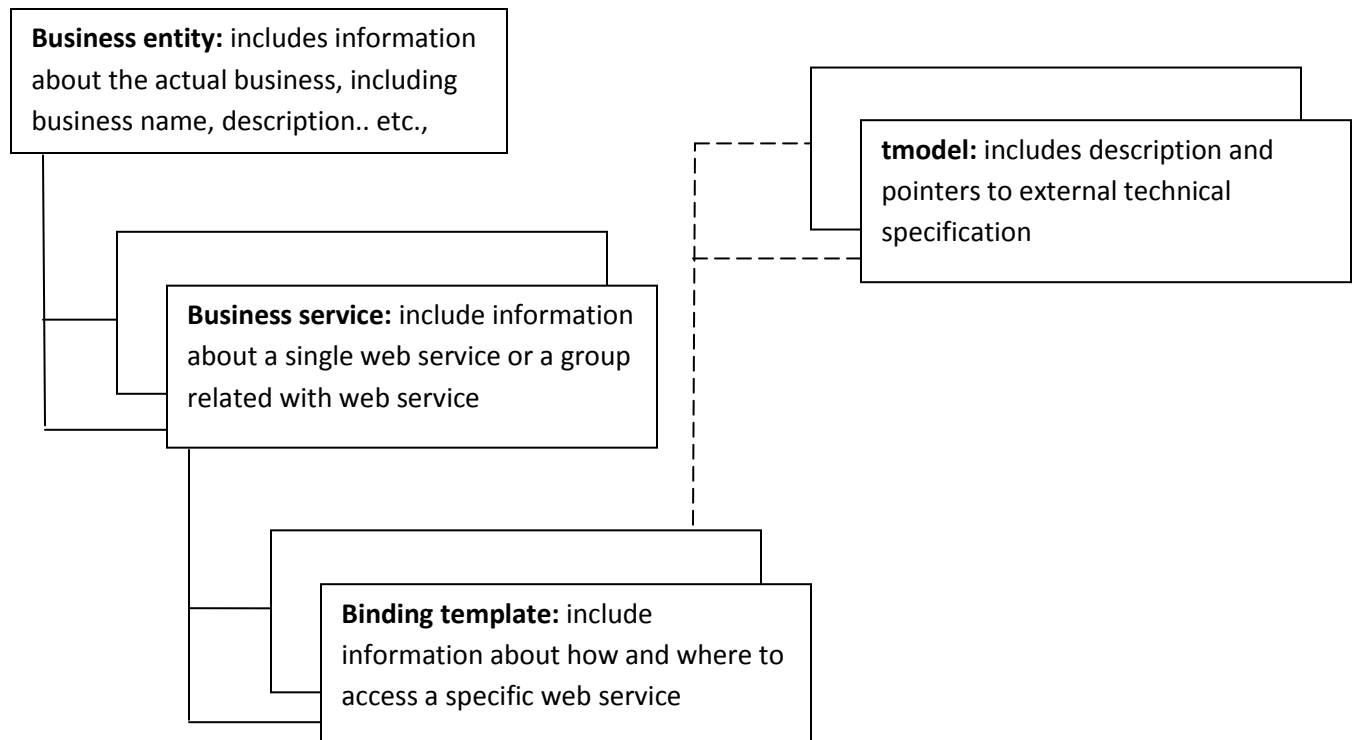
### **GREEN PAGE:**

It includes technical information about the web service.

White and yellow pages directories were, earlier, available in the printed format.

In the present one, these directories are available online and are accessible globally.

### **UDDI DATA MODEL:**



## **BUSINESS ENTITY:**

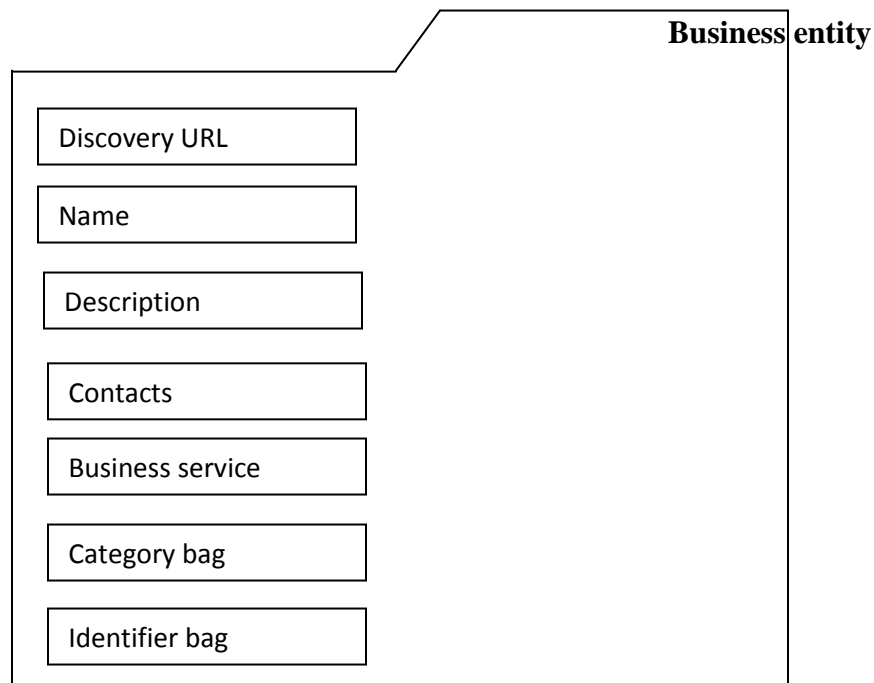
Business entity data structure represents the top-level UDDI element. This element holds descriptive information about business.

The business entity data structure contains information such as contact information, categorization, identifiers, descriptions and business relations.

The business entity data structure defines an important attribute called business key, which is a unique key.

This key is provided by organization itself at the time of registration or it could be generated by the registry while publishing the service. Information retrieval from the UDDI registry will contains the key information.

The graphical representation of the business entity data structure.



The <identifier bag> data structure contains a list of identifiers; the <category bag> data structure contains a list of business categories that describe a specific business aspect.

```
<businessentity businesskey="....." operator="....." authorizedname=".....">
 <name>.....</name>
 <description>..... </description>
 <contact>..... </contact>
 <businessservice>.....</businessservice>
 <categorybag>
```

```

 <keyedReference tmodelkey="...." Keyname="....."
keyvalue="....."/>
 </categorybag>
 <identifierbag>
 <keyedReference tmodelkey="...." Keyname="....."
keyvalue="....."/>
 </identifierbag>
</businessentity>

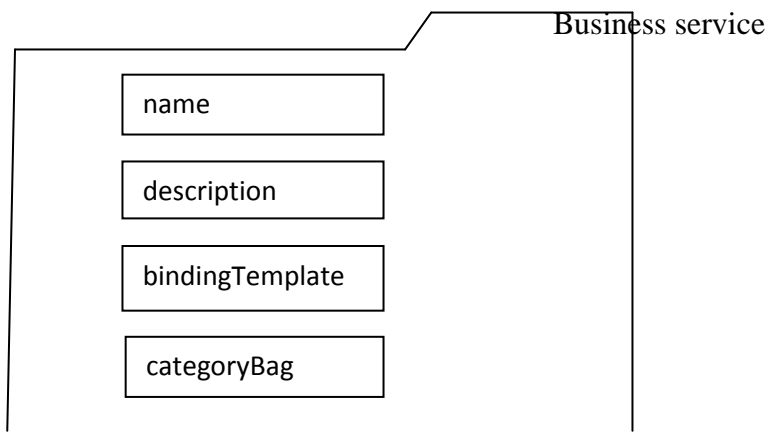
```

### **Business Service:**

Business service data structure provides information about a single web service or group related with web services.

The business service data structure contains descriptive information such as the name of the service, description, classification.

The graphical representation of the business service is,



The data structure defines two important attributes, business key and service key.

```

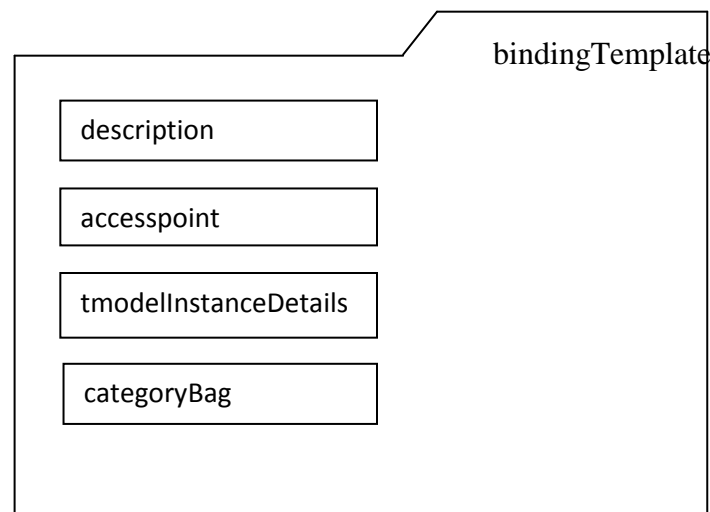
<businessService
 servicekey=" "
 businessKey=" ">
 <name> </name>
 <description> </description>
 <bindingTemplates> </bindingTemplates>
 <categoryBag> </categoryBag>
</businessService>

```

### **Binding Template:**

- Binding template data structure represents individual web service.

- The data structure stores a list of technical information needed by the application to find and interact with web service being described.
- The data structure defines two important data structure **accesspoint** and **tmodelInstanceDetails**
- The accespoint is string that is used to represent a suitable network address for invocation of the web service being described. This could be a URL, email address or any other textual information.
- The accesspoint supports an optional attribute called UseType.
- UDDI provides four pre-defined UseType attributes value, endpoint, bindingTemplate, hostingRedirector and wsdlDeployment.
- Among all these, wsdlDeployment is important for the understanding of web services.
- This includes that the accesspoint data structure points to WSDL document that contain necessary information for binding and service invocation.
- The tmodelInstanceDetails data structure is a mandatory structure. This data structure hosts another important data structure called tmodelInstanceInfo.
- This tmodelInstanceInfo element defines on attribute called tmodelKey.
- This is a mandatory attribute that represents a specification with which the web service that is represented by the corresponding bindingTemplate compiles.



```

<bindingTemplates>
 <bindingTemplate
 serviceKey=" "
 bindingKey=" "
 <description> </description>
 <accessPoint URLType= > </accessPoint>
 <tmodelInstanceDetails>
 <tmodelInstanceInfo tmodelKey=" " </>

```



```
</tmodelInstanceDetails>
</bindingTemplate>
</bindingTemplates>
```

### **tModel:**

- tModel stands for technical model.
- tModel are primarily used to provide pointers to external technical specifications.
- These data structures help the application realize the required technical information about the web services.
- The tModel data structure consists of a <name>, <description> and <overviewDoc> elements.
- The tModel data structure defines an optional attribute called tModelKey.
- The given tModel is uniquely identified by a tModelKey.
- The tModel sometimes is referred to as the ‘digital fingerprint’ for determining the specifics of interacting with a particular web service.

```
<tmodel tmodelkey="....." operator="....." Authorizedname=".....">
 <name>.....</name>
 <description>.....</description>
 <overviewdoc>
 <overviewurl>.....</overviewurl>
 </overviewdoc>
 <categorybag>
 <keyedreference tmodelkey="....." keyname="....." keyvalue="....."/>
 </categorybag>
</tmodel>
</business entity>
```

### **UDDI REGISTRIES:**

#### **NODE API SETS:**

There is the API that is used for inter-UDDI communication implementation. The purpose is to manipulate the data stored in different UDDI implementations.

The node API sets are available for implementing a variety of tasks such as UDDI inquiry, UDDI publications, UDDI subscription and UDDI security.

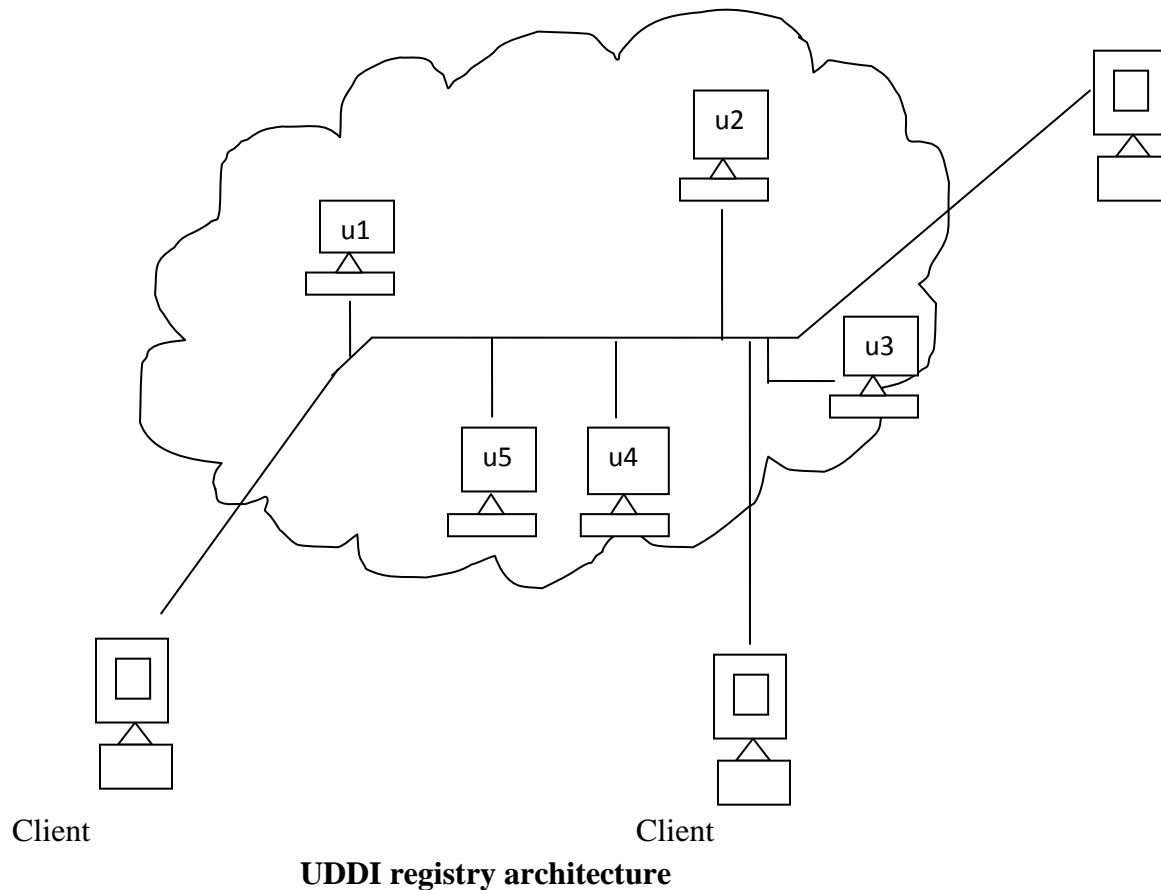
#### **UDDI NODE:**

A system that hosts an application to support at least one of the node API sets is called a UDDI node. A UDDI node must ensure the following.

1. Interaction with the UDDI data through API sets

2. Membership with exactly one registry
3. Ability to access and manipulate the UDDI data

One or more UDDI nodes may be combined to form a UDDI registry. These UDDI nodes are designed to collectively manage the business data in the registry. The UDDI registry architecture is shown in the figure.



### **Service publication:**

Service publication involves creating, updating and removing information related to a company, business and/or services on the UDDI business registry.

The task of service publication is an administration activity. Normal users are not authorized for doing this activity and access to such activity will be granted through authentication and authorization mechanism.

The service publication needs to be brought about using service-publishing applications. These applications communicate with the registries using soap. Service-publishing applications use appropriate API's. That helps in generating the publishing request based on UDDI specification.

### **Service creation and modification:**

Messages are used for creating, modifying and deleting of UDDI information. There are authenticated operations. This is achieved through publisher's interfaces. Four soap messages are used to save each of the four data structure.

They are save Business, save Service, save Binding and save tmodel.

#### **save Business:**

This operation is represented using the <save\_business> element. This stores or updates the information contained within the <businessEntity> element.

#### **save Service:**

This operation inserts or modifies the information provided inside the <businessService> data structures. This operation will affect the <bindingTemplate> and the <tmodel> data structure.

#### **save Binding:**

This operation will insert or modify the information provided in the <bindingTemplate> data structure. The save Binding also takes care of tmodel data structure.

#### **save tmodel:**

This operation inserts or updates the <tmodel> data structure information.

#### **Service Deletion:**

Similarly, we have delete messages for each of the four data structures. These delete operations are authenticated operations. They are Delete Business, Delete Service, Delete Binding and Delete tmodel.

#### **Delete Business:**

Deletion corresponds to the deletion of existing information requires UUID. UUID stands for Universal Unique Identifier. UUID's are assigned when data structures are first created and inserted into the UDDI business registry.

UUID are hexadecimal strings, and they are in accordance with the algorithm defined by ISO. It is an authentication operation. <delete\_business> element is used to provide the information deletion.

#### **Delete Service:**

The element <delete\_Service> is used to encode the service data for deletion purpose. It requires valid UUID for deleting the business service data.

Delete Binding and Delete tmodel are authenticated operations, which accept valid UUID's corresponding to binding and tmodel informations.

### **Service Discovery:**

Service discovery will enable clients to search and find out the details of the company, services and other related informations from the UDDI business registry.

For searching we use the inquiry interface. Searching can be carried out at two different levels

- Browse
- Drill down

The service discovery needs to be brought about using service\_searching or service querying applications. These applications communicate with the registries using soap.

Service\_querying applications, use appropriate API's that help in generating appropriate soap based queries based on UDDI specifications.

### **Information Browsing:**

Browsing refers to a superficial search of the UDDI Business registry. Within browsing, we have four different categories. Find Business, Find Service, Find Binding and Find tmodel.

All the find and drill down operations are non-authenticated operations.

### **Find Business:**

Find Business is a generic query operation. In order to search for business <find\_business> element is used to locate information about one or more business.

### **Find Service:**

Find Service is once again a general-purpose query.

### **Web service policy:**

Web service policy provides a flexible and extensible grammar for expressing the capabilities, requirements and general characteristics of entities in a XML webservice- based system.

WS-policy defines a policy; each policy is a collection of policy alternatives. A policy alternative is a collection of policy assertions. A policy assertion represents an individual requirement, capability or other properties of a behavior.

**Eg:** Authentication schema, transport protocol selection, privacy policy, QOS characteristics.

## Goals:

The goal of WS-policy is to provide the mechanism needed to enable webservice applications to specify policy information.

## Eg:

1. <wsp:Policy xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
2. <wsp:ExactlyOne>
3. <sp:Basic256 Rsa 15/>
4. <sp:TripleDes Rsa 15/>
5. </wsp:ExactlyOne>
6. </wsp:Policy>

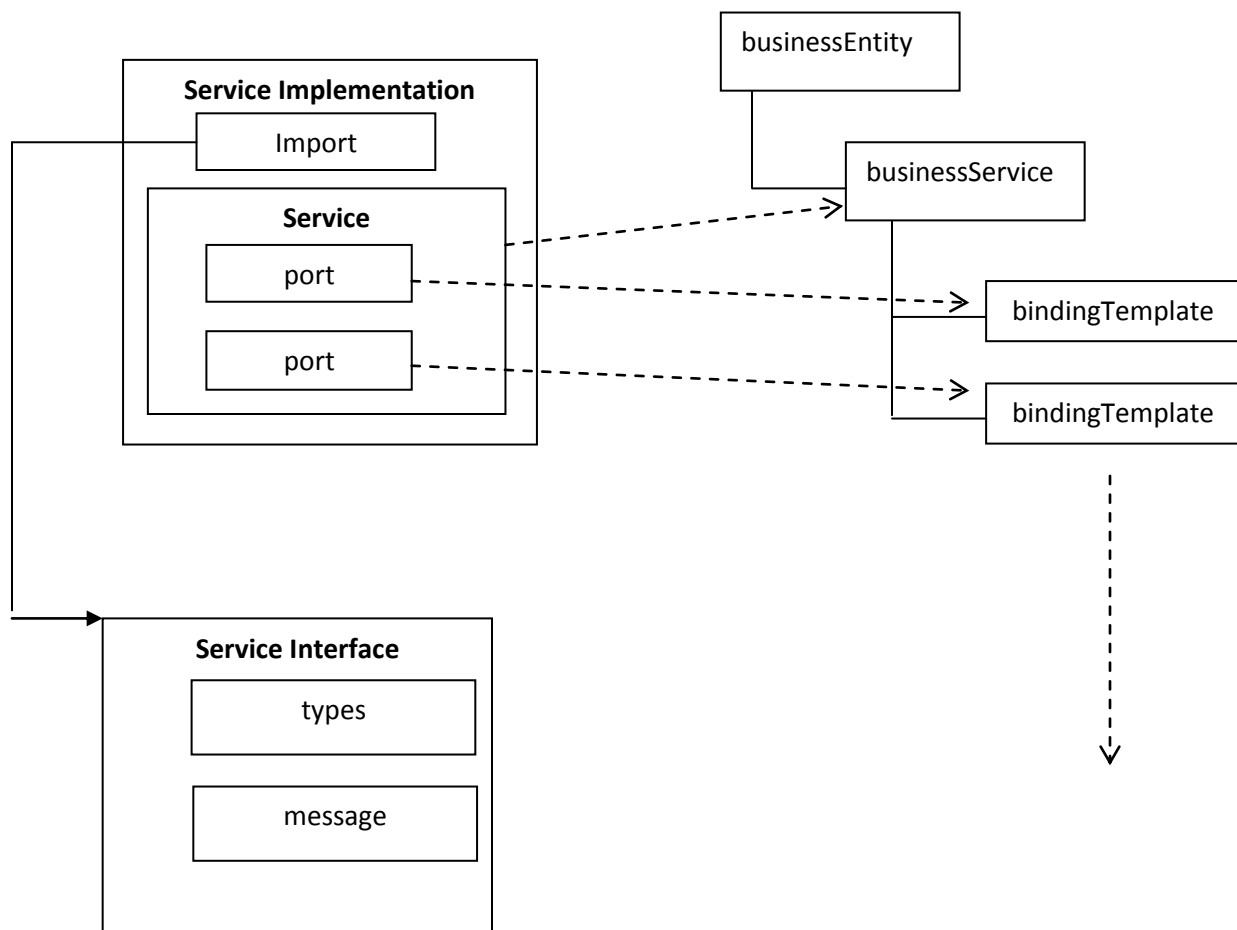
Line (1-6) represents a policy for the algorithm.

Line (2-5) illustrates the exactly one policy operator.

Policy operators group policy assertions into policy alternatives.

## Publishing and Finding WSDL Description in a UDDI registry:

In order to understand how UDDI uses WSDL as a description language, you must be clear on how WSDL documents are mapped to the UDDI structure. The implementation document maps to the UDDI business service element whereas the service interface. The first step is publishing a WSDL description in UDDI registry is publishing the service interface as a tModel.





- Set the **name** field of the **tmodel** to the target Namespace attribute of the definitions element **binding** element. This field is used to locate the appropriate tmodel.
- The description field of the tmodel corresponds to the documentation element of the interface document. This field can have a maximum of 256 characters.
- Set the overviewURL field of the tmodel to the URL and binding specification in the interface document.
- Set the categoryBag field of the tmodel so that it's keyed reference in uddi-org: types and its keyVal in wsdlSpec. This defines the UDDI entry as a WSDL service interface definition.

### **tModel.xml—A tModel Created from a WSDL Service Interface**

```

<? xml version="1.0"?>
<tModel tModelKey="">
<name>http://www.myclassservice.com/MyClass-interface</name>
<description xml: lang="en">
 Service interface definition for our demo service.
</description>
<overviewDoc>
<description xml: lang="en">
WSDL Service Interface Document
</description>
<overviewURL>
http://www.myclassservice.com/MyClass-interface#MyClass_serviceBinding
</overviewURL>
</overviewDoc>
<categoryBag>
<keyedReference tModelKey="UUID:C1ACF26D-9672-44049D70-889DFDA9D9F8"
keyName="uddi-org:types" keyValue="wsdlSpec"/>
<keyedReference tModelKey="UUID:C1ACF26d-9672-4404-9D70-889DFDA9D9F8"
keyName="uddi-org:types" keyValue="wsdlSpec"/>

<keyedReference tModelkey="UUID:DB77450D-9FA8-45D4-A7BC-9C8C7D998F8D"
keyName="Sample Web Service" keyValue="12345678"/>
</categoryBag>
</tModel>

```

Next, we must create the businessService and bindingTemplate elements that correspond to the WSDL service implementation document.

**The businessService has the following fields:**

- The name field of the businessService is set to the name attribute of the service element in the implementation document.
- The description field of the businessService comes from the documentation element in the implementation document.

**The bindingTemplate has the following fields:**

- The description field of the bindingTemplate comes from the first 256 characters of the documentation element of the port element, if it exists.
- The accesspoint field is set to the value of the location attribute of the extension element that is associated with the port element, in the case of a SOAP or HTTP binding.
- The bindingTemplate contains one tModelInstanceInfo field for each tModel it references.
- The overviewURL field directly references the service implementation document; however, it is only used to provide human readability.

Putting these instructions together, we can build the UDDI businessService that corresponds to the WSDL interface document. The businessService appears as follows

businessService.xml – businessService and bindingTemplate elements created from a WSDL Service Implementation

```
<businessService businessKey="..." serviceKey="...">
<name>MyClass_Service</name>
<description xml:lang="en">
 IBM WSTK V2.4 generated service definition file
</description>
<bindingTemplates>
<bindingTemplate bindingKey="..." serviceKey="...">
<description></description>
<accesspoint URLType="http">
http://localhost:8080/soap/servlet/rpcrouter
</accesspoint>
<tModelInstanceDetails>
<tModelInstanceInfo tModelKey="[tModel key for service interface]">
<instanceDetails>
<overviewURL>
```

[http://localhost:8080/wsdl/MyClass\\_Service.interface.wsdl](http://localhost:8080/wsdl/MyClass_Service.interface.wsdl)

</overviewURL>

</instanceDetails>

</tModelInstanceInfo>

</tModelInstanceDetails>

</bindingTemplate>

</bindingTemplates>

<categoryBag>

<keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-3663DA8D8CFB"  
keyName="Sample Web Service" keyValue="84121801"/>

</categoryBag>

</businessService>

## **The UDDI API:**

### **Inquiry API:**

The inquiry queries are as follows:

- **find\_binding:** locates specific binding within a registered businessService and returns a bindingDetail message that contains a bindingTemplate elements structre. If there are no matches, the returned bindingDetail will be empty. If an error occurs, a dispositionReport structure will be returned in a SOAP Fault element.
- **find\_business:** locates information about one or more business and returns a businessList message. Searches can be performed on name elements (or partial name elements), identifierBag elements, categoryBag elements, tModelBag elements, or discoveryURL elements. tModelBag elements are collection of tModel elements that allow searches for compatible bindings. If there are no matches, an empty businessList is returned. Errors are handled as they are with the find\_binding query.
- **find\_service:** locates specific services within a registered businessEntity and returns a serviceList message. Searches can be performed on name elements (or partial name elements), identifierBag elements, categoryBag elements, or tModelBag elements. If there are no matches, an empty businessService structure is returned. Errors are handled as they are with the find\_binding query.
- **find\_tModel:** locates one or more tModel information structures and returns a tModelList structure, which is a list of abbreviated information about tModel elements that match the search criteria. Search parameters, no match conditions, and error conditions are handled the same as the preceding queries.

### **Publication API:**



The publication queries are as follows

- `delete_binding`: removes an existing `bindingTemplate` from the `bindingTemplates` collection that is part of a specified `businessService` structure. The `bindingTemplate` is identified by its `bindingKey`. If successful, a `dispositionReport` with a single success indicator is returned.
- `delete_business`: deletes a business by deleting registered `businessEntity` structures from the registry. The `businessEntity` is identified by its `businessKey`.
- `delete_service`: deletes an existing `businessService` structure from the `businessServices` collection, which is part of a specified `businessEntity`. The `businessService` is identified by its `serviceKey`.
- `delete_tModel`: used to delete registered information about one or more `tModel` structures. If there are any references to a `tModel` when this call is made, the `tModel` will be marked as deleted, or hidden, instead of being physically removed. Hidden `tModel` elements can still be accessed by the owner but are not returned in search results. However, the details in a hidden `tModel` are still accessible, so this should be nulled out with the `save_tModel` call, if the owner wishes the details to be deleted.

#### **References:**

- Leon Shklar, Rich Rosen, "Web Application Architecture: Principles, Protocols, and Practices", John Wiley & Sons, Second Edition.
- <http://www.w3schools.com/>