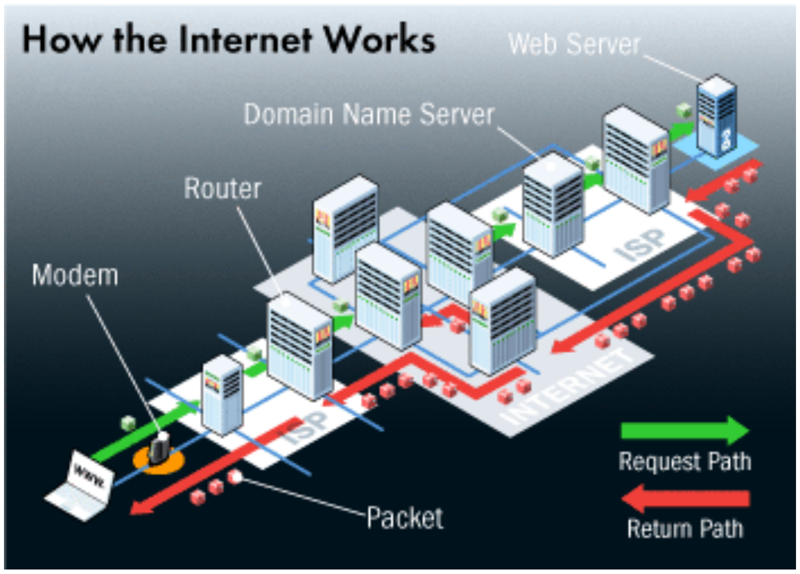
Chapter 1 – Introduction

CLIENT SERVER ARCHITECTURE

The client–server model is an approach to computer network programming developed at Xerox PARC during the 70s. It is now prevalent in computer networks. Email, the World Wide Web, and network printing all apply the client–server model.

The model assigns one of two roles to the computers in a network: Client or server. A server is a computer system that selectively shares its resources; a client is a computer or computer program that initiates contact with a server in order to make use of a resource. Data, CPUs, printers, and data storage devices are some examples of resources. This sharing of computer resources is called time-sharing, because it allows multiple people to use a computer (in this case, the server) at the same time. Because a computer does a limited amount of work at any moment, a time-sharing system must quickly prioritize its tasks to accommodate the clients.

Clients and servers exchange messages in a request-response messaging pattern: The client sends a request, and the server returns a response. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. Whether a computer is a client, a server, or both, it can serve multiple functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software on the same computer. Communication between servers, such as to synchronize data, is sometimes called inter-server or server-to-server communication. ***- Wiki***



A **web browser** or **Internet browser** or **Web Client** is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. An *information resource* is identified by a Uniform Resource Identifier (URI) and may be a web page, image, video, or other piece of content Hyperlinks present in resources enable users to easily navigate their browsers to related resources.

In order of release:

- Netscape Navigator and Netscape Communicator, October 13, 1994
- Internet Explorer 1, August 16, 1995
- Opera, 1996
- Mozilla Navigator, June 5, 2002
- Safari, January 7, 2003
- Mozilla Firefox, November 9, 2004
- Google Chrome, September 2, 2008

Web Servers

Web servers are computers that deliver (*serves up*) Web pages. Every Web server has an IP address and possibly a domain name. Web servers are computers on the Internet that host websites, serving pages to viewers upon request. This service is referred to as web hosting. Every web server has a unique address so that other computers connected to the Internet know where to find it. The Internet Protocol (IP) address looks something like this: 69.93.141.146. This address maps to a more human friendly address, such as http://www.   .np. Web hosts rent out space on their web servers to people or businesses to set up their own websites. The web server allocates a unique website address to each website it hosts.

For example, if you enter the URL *http://www.mechicampus.com.np/index.html* in your browser, this sends a request to the Web server whose domain name is mechicampus.com.np The server then fetches the page named *index.html* and sends it to your browser.

Any computer can be turned into a Web server by installing server software and connecting the machine to the Internet. Two leading Web servers are Apache (the most widely installed Web server), and Microsoft's Internet Information Server (IIS).

Apache HTTP Server **FREE & Open Source**

Apache HTTP Server (also referred to as simply "Apache") has, at the time of writing, been the most popular web server on the web since 1996. Apache is developed and maintained by the Apache Software Foundation, which consists of a decentralized team of developers. The software is produced under the Apache license, which makes it free and open source. Apache is available for a range of operating systems, including Unix, Linux, Novell Netware, Windows, Mac OS X, Solaris, and FreeBSD.

Nginx FREE & Open Source

Nginx (pronounced "engine X") is the second most popular open source web server currently on the Internet. Though development only started in 2002, its currently used by over 6% of web domains. It is a lightweight HTTP server, and can also serve as a reverse proxy and IMAP/POP3 proxy server. It's licensed under a BSD-like license. It runs on UNIX, GNU/Linux, BSD, Mac OS X, Solaris, and Windows.

Nginx was built with performance in mind, in particular to handle ten thousand clients simultaneously. Instead of using threads to handle requests, like traditional servers, Nginx uses an event-driven (asynchronous) architecture. Its more scalable and uses less, and more predictable, amounts of memory. In addition to the basic HTTP features, Nginx also supports name-based and IP-based virtual servers, keep-alive and pipelined connections, and FLV streaming. It can also be reconfigured and upgraded online without interruption of the client processing.

Lighttpd FREE & Open Source

Lighttpd (pronounced "lighty") is the third most popular open source web server. This lightweight server was initially released in 2003 and currently serves less than 1% of web domains. It's licensed under a revised BSD license and runs on Unix and Linux.

Like nginux, lighttpd is a lightweight server built for performance with a goal of handling ten thousand clients simultaneously. It also uses an event-driven (asynchronous) architecture.

Tornado FREE & Open Source

Tornado is a scalable non-blocking web server, meant to be used for real-time web services. According to the website, it is able to handle thousands of simultaneous standing connections. It has built-in cross-site request forgery (or XSRF) protection, aggressive file caching (to improve performance), support for user interface (UI) modules, etc. The server is available in source form and requires Python to be installed.

Microsoft Personal Web Server (**PWS**) **(Proprietary)** is a scaled-down web server software for Windows operating systems. It has fewer features than Microsoft's Internet Information Services (IIS) and its functions have been superseded by IIS and Visual Studio. Microsoft officially supports PWS on Windows 95, Windows 98, Windows 98 SE, and Windows NT 4.0. Prior to the release of Windows 2000, PWS was available as a free download as well as included on the Windows distribution CDs. PWS 4.0 was the last version and it can be found on the Windows 98 CD and the Windows NT 4.0 Option Pack.

Microsoft Internet Information Services (IIS) **(Proprietary)**

IIS is, at the time of writing, the second most popular web server on the web. It is however, gaining market share, and if the current trend continues, it won't be long before it overtakes Apache.

IIS comes as an optional component of most Windows operating systems. You can install IIS by using *Add/Remove Windows Components* from *Add or Remove Programs* in the Control Panel.

DNS

DNS is a protocol within the set of standards for how computers exchange data on the Internet and on many private networks, known as the TCP/IP protocol suite. Its basic job is to turn a user-friendly **domain name** like "mechicampus.edu.np" into an Internet Protocol (IP) address like 78.42.245.48 that computers use to identify each other on the network. It's like your computer's GPS for the Internet.

Computers and other network devices on the Internet use an IP address to route your request to the site we're trying to reach. This is similar to dialing a phone number to connect to the person. We don't have to keep address book of IP addresses. Instead, we just connect through a **domain name server**, also called a **DNS server** or **name server**, which manages a massive database that maps domain names to IP addresses.

Whether we are accessing a Web site or sending e-mail, computer uses a DNS server to look up the domain name. The proper term for this process is **DNS name resolution**, and the DNS server resolves the domain name to the IP address. For example, when you enter "http://www.mechicampus.edul.np" in your browser, part of the network connection includes resolving the domain name "mechicampus.edu.np" into an IP address, like 78.42.245.48, for it's Web servers.

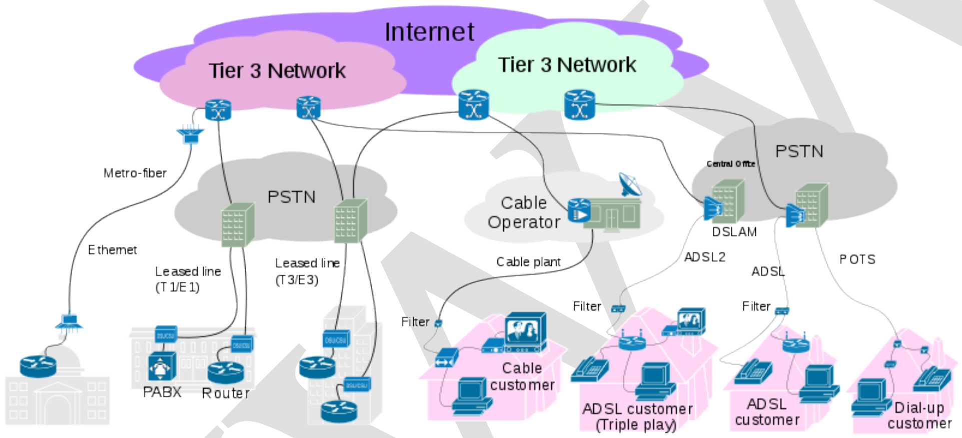
We can always bypass a DNS lookup by entering 78.42.245.48 directly in your browser. However, we like to remember "mechicampus.edu.np" instead of its IP. A Web site's IP address can change over time, and some sites associate multiple IP addresses with a single domain name.

Without DNS servers, the Internet would shut down very quickly. Typically, when we connect to our home network, Internet service provider (ISP) or WiFi network, the modem or router that assigns computer's network address also sends some important network configuration information to our computer or mobile device. That configuration includes one or more DNS servers that the device should use when translating DNS names to IP address. ***- HSW***

ISP

An Internet service provider (ISP) is an organization that provides access to the Internet. ISPs directly connect clients to the Internet using copper wires, wireless or fiber-optic connections.

- ISP stands for **Internet Service Provider**.
- An ISP provides **Internet Services**.
- A common Internet service is **web hosting**.
- Web hosting means **storing your web site on a public server**.
- Web hosting normally includes **email services**.
- Web hosting often includes **domain name registration**.



WEB HOSTING

A web hosting service is a type of Internet hosting service that allows individuals and organizations to make their website accessible via the World Wide Web. Web hosts are companies that provide space on a server owned or leased for use by clients, as well as providing Internet connectivity, typically in a data center. Web hosts can also provide data center space and connectivity to the Internet for other servers located in their data center, called colocation, also known as Housing in Latin America or France. The scope of web hosting services varies greatly.

The most basic is web page and small-scale file hosting, where files can be uploaded via File Transfer Protocol (FTP) or a Web interface. The files are usually delivered to the Web "as is" or with minimal processing.

Many Internet service providers (ISPs) offer this service free to subscribers. Individuals and organizations may also obtain Web page hosting from alternative service providers. Personal web site hosting is typically free, advertisement-sponsored, or inexpensive. Business web site hosting often has a higher expense. ***- Wiki***

Hosting can be FREE, SHARED or DEDICATED.

Web Hosting - http://goo.gl/2bSjX

FTP & Its Types

***F**ile **T**ransfer **P**rotocol,* the protocol for exchanging files over the Internet. FTP works in the same way as HTTP. FTP uses the Internet's TCP/IP protocols to enable data transfer. FTP is most commonly used to download a file from a server using the Internet or to upload a file to a server (e.g., uploading a Web page file to a server).

- File Transfer Protocol (FTP) is a method of transferring files from a client to a server or vice versa.
- Files are transferred over the Internet using TCP/IP protocol.
- FTP is old-time protocol that maintains two simultaneous connections.
- The first connection uses the telnet remote login protocol to log the client into an account and process commands via the *protocol interpreter*.
- The second connection is used for the *data transfer process*.
- Whereas the first connection is maintained throughout the FTP session, the second connection is opened and closed for each file transfer.
- The FTP protocol also enables an FTP client to establish connections with two servers and to act as the third-party agent in transferring files between the two servers.
- FTP servers rarely change, but new FTP clients appear on a regular basis.
- Although FTP is a command-line oriented protocol, the new generation of FTP clients hides this orientation under a GUI environment.
- A client makes a TCP connection to the server's port 21. This connection, called the *control connection*, remains open for the duration of the session, with a second connection, called the *data connection*, either opened by the server from its port 20 to a negotiated client port or opened by the client from an arbitrary port to a negotiated server port as required to transfer file data.

[FTP is a TCP based service exclusively. There is no UDP component to FTP. FTP is an unusual service in that it utilizes two ports, a 'data' port and a 'command' port (also known as the control port). Traditionally these are port 21 for the command port and port 20 for the data port. The confusion begins however, when we find that depending on the mode, the data port is not always on port 20.]

Types of FTP

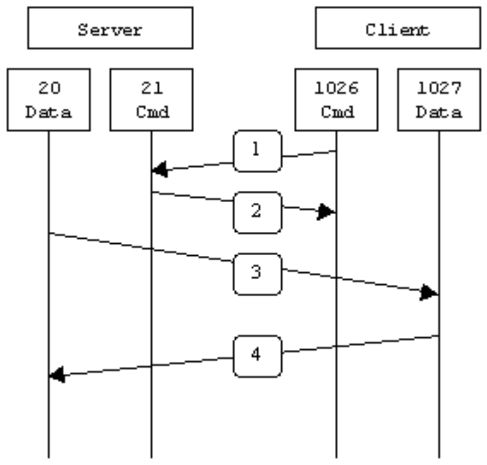
- Active FTP

In active mode FTP the client connects from a random unprivileged port (N > 1023) to the FTP server's command port, port 21. Then, the client starts listening to port N+1 and sends the FTP command PORT N+1 to the FTP server. The server will then connect back to the client's specified data port from its local data port, which is port 20.

From the server-side firewall's standpoint, to support active mode FTP the following communication channels need to be opened:

- FTP server's port 21 from anywhere (Client initiates connection)
- FTP server's port 21 to ports > 1023 (Server responds to client's control port)
- FTP server's port 20 to ports > 1023 (Server initiates data connection to client's data port)
- FTP server's port 20 from ports > 1023 (Client sends ACKs to server's data port)

When drawn out, the connection appears as follows:



- In step 1, the client's command port contacts the server's command port and sends the command PORT 1027.
- The server then sends an ACK back to the client's command port in step 2.
- In step 3 the server initiates a connection on its local data port to the data port the client specified earlier.
- Finally, the client sends an ACK back as shown in step 4.

The main problem with active mode FTP actually falls on the client side. The FTP client doesn't make the actual connection to the data port of the server - it simply tells the server what port it is listening on and the server connects back to the specified port on the client. From the client side firewall this appears to be an outside system initiating a connection to an internal client - something that is usually blocked.

- Passive FTP

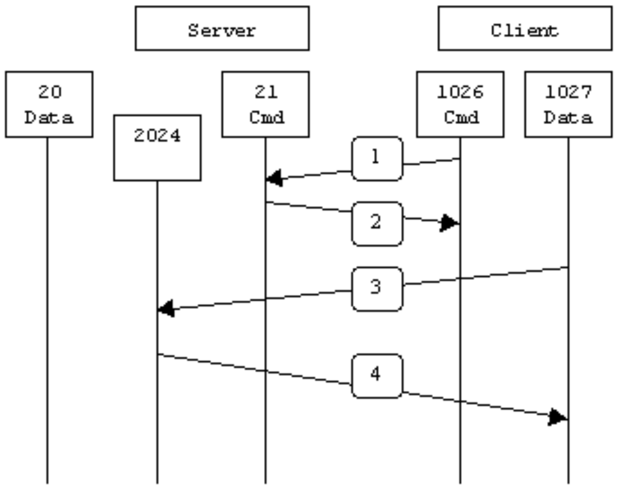
In order to resolve the issue of the server initiating the connection to the client a different method for FTP connections was developed. This was known as passive mode, or PASV, after the command used by the client to tell the server it is in passive mode.

In passive mode FTP the client initiates both connections to the server, solving the problem of firewalls filtering the incoming data port connection to the client from the server. When opening an FTP connection, the client opens two random unprivileged ports locally (N > 1023 and N+1). The first port contacts the server on port 21, but instead of then issuing a PORT command and allowing the server to connect back to its data port, the client will issue the PASV command. The result of this is that the server then opens a random unprivileged port (P > 1023) and sends the PORT P command back to the client. The client then initiates the connection from port N+1 to port P on the server to transfer data.

From the server-side firewall's standpoint, to support passive mode FTP the following communication channels need to be opened:

- FTP server's port 21 from anywhere (Client initiates connection)
- FTP server's port 21 to ports > 1023 (Server responds to client's control port)
- FTP server's ports > 1023 from anywhere (Client initiates data connection to random port specified by server)
- FTP server's ports > 1023 to remote ports > 1023 (Server sends ACKs (and data) to client's data port)

When drawn, a passive mode FTP connection looks like this:



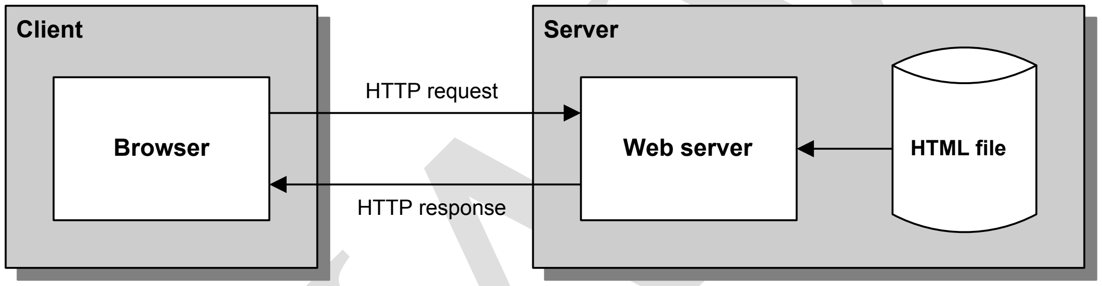
- In step 1, the client contacts the server on the command port and issues the PASV command.
- The server then replies in step 2 with PORT 2024, telling the client which port it is listening to for the data connection.
- In step 3 the client then initiates the data connection from its data port to the specified server data port. Finally, the server sends back an ACK in step 4 to the client's data port.

- While passive mode FTP solves many of the problems from the client side, it opens up a whole range of problems on the server side. The biggest issue is the need to allow any remote connection to high numbered ports on the server. Fortunately, many FTP daemons, including the popular WU-FTPD allow the administrator to specify a range of ports, which the FTP server will use. The second issue involves supporting and troubleshooting clients, which do (or do not) support passive mode. FTP Clients like FileZilla give user option to select the FTP mode.

With the massive popularity of the World Wide Web, many people prefer to use their web browser as an FTP client. Most browsers only support passive mode when accessing ftp:// URLs. This can either be good or bad depending on what the servers and firewalls are configured to support.

More Reading…

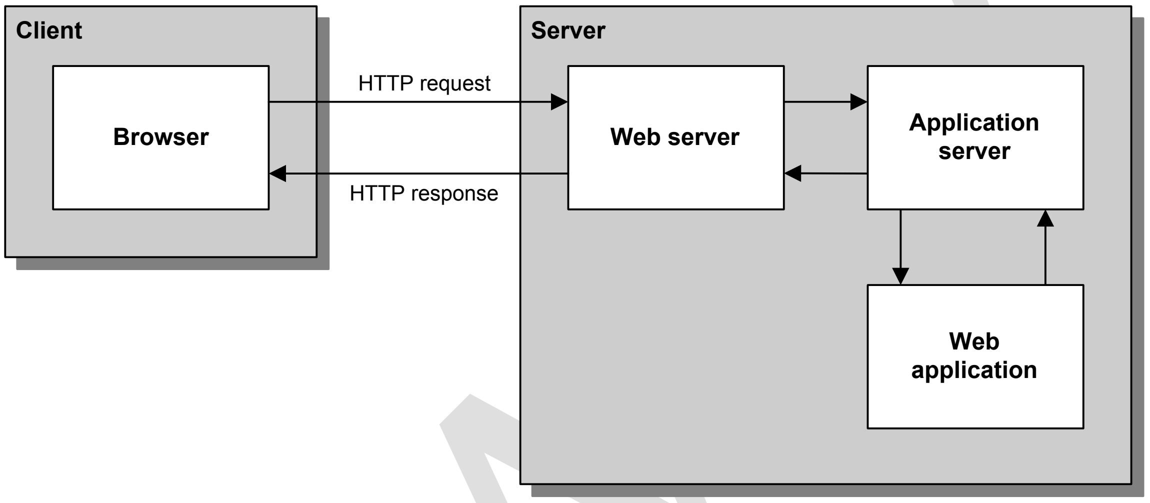
Static web page



- A *static web page* is an HTML document that is the same each time it's viewed. It doesn't change in response to user input.
- Static web pages are usually simple HTML files that are stored on the web server with a file extension of .htm or .html. When a browser requests a static web page, the web server retrieves the file from disk and sends it back to the browser.
- A web browser requests a page from a web server by sending the server an HTTP message known as an HTTP *request*.
- The HTTP request includes, among other things, the name of the HTML file being requested and the Internet address of both the browser and the web server.
- A user working with a browser can initiate an HTTP request in several ways. One way is to type the address of a web page, called a *URL*, or *Uniform Resource Locator*, into the browser's address area and press Enter. Another way is to click a link that refers to a web page.

- A web server replies to an HTTP request by sending a message known as an *HTTP response* back to the browser.
- The HTTP response contains the addresses of the browser and the server as well as the HTML document that's being returned.

Dynamic web page



- A *dynamic web page* is an HTML document that's generated by a web application. Often, the web page changes according to information that's sent to the web application by the browser.
- When a web server receives a request for a dynamic web page, the server passes the request to an *application server*.
- The application server executes the web application, which generates an HTML document. This document is returned to the application server, which passes it back to the web server. The web server, in turn, sends the document back to the browser.
- After the page is displayed, the user can interact with it using its controls. Some of those controls let the user *post* the page back to the server, so it's processed again using the data the user entered.
- Each application mapping specifies which application should be run to process files with that extension.
- If the file extension isn't in the list of application mappings, the requested file is returned to the browser without any processing.
- The process that begins with the user requesting a web page and ends with the server sending a response back to the client is called a *round trip*.

- After a web application generates an HTML document, it ends. Then, unless the data the application contains is specifically saved, that data is lost.

HTTP Overview

- HTTP, Hyper Text transfer Protocol is the standard Web transfer protocol.
- The HTTP is the language that Web clients and Web servers use to communicate with each other.
- It is essentially the backbone of the web.
- It is constantly evolving protocol with several versions in use and others are still under development.
- This protocol has two items: the set of requests from browsers to servers and the set of responses going back the other way.
- It is a stateless protocol and does not maintain any information from one transaction to the next, so the next transaction needs to start all over again
- The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there's no additional overhead for tracking sessions from one connection to the next.
- Default Port used by HTTP is 80.

HTTPS

- HTTPS stands for Hyper Text Transfer Protocol Secure. It is a secured version of the HTTP protocol that uses SSL (protocol primarily developed with secure, safe Internet transactions in mind as the encryption layer. SSL layer also offers strong authentication methods.
- HTTPS allows secure ecommerce transactions, such as online banking.
- HTTPS connects on port 443, while HTTP is on port 80
- HTTPS encrypts the data sent and received with SSL, while HTTP sends it all as plain text.
- When a user connects to a website via HTTPS, the website encrypts the session with a digital certificate. A user can tell if they are connected to a secure website if the website URL begins with https:// instead of http://.
- Secure Sockets Layer uses a cryptographic system that encrypts data with two keys.
- SSL transactions are negotiated by means of a key based encryption algorithm between the client and the server, this key is usually either 40 or 128 bits in strength (the higher the number of bits the more secure the transaction).
- When a SSL Digital Certificate is installed on a web site, users can see a padlock icon at the bottom area of the navigator. When an Extended Validation Certificates is installed on a web site, users with the latest versions of Firefox, Internet Explorer or Opera will see the green address bar at the URL area of the navigator.

[***Note***: HTTPS should not be confused with S-HTTP, a security-enhanced version of HTTP. SSL and S-HTTP have very different designs and goals so it is possible to use the two protocols together. Whereas SSL is designed to establish a secure connection between two computers, S-HTTP is designed to send individual messages securely.]

Why Is A SSL Certificate Required?

With booming Internet trends and fraud, most will not submit their private details on the web unless they know that the information they provide is securely transmitted and not accessible for anyone to view.

HTTP Transaction

All HTTP transactions follow the same general format. Each client request and server response has three parts: the request or response line, a header section, and the entity body. ***The client initiates a transaction as follows:***

1. The client contacts the server at a designated port number (by default, 80). Then it sends a document request by specifying an HTTP command called a *method*, followed by a document address, and an HTTP version number. For example:

```
GET /index.html HTTP/1.0
```

Uses the GET method to request the document *index.html* using version 1.0 of HTTP.

2. Next, the client sends optional header information to inform the server of its configuration and the document formats it will accept. All header information is given line by line, each with a header name and value. For example, this header information sent by the client indicates its name and version number and specifies several document preferences:

```
User-Agent: Mozilla/2.02Gold (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

The client sends a blank line to end the header.

3. After sending the request and headers, the client may send additional data. CGI programs using the POST method mostly use this data. Clients like Netscape Navigator-Gold, to publish an edited page back onto the Web server, may also use it.

The server responds in the following way to the client's request:

1. The server replies with a status line containing three fields: HTTP version, status code, and description. The HTTP version indicates the version of HTTP that the server is using to respond. The status code is a three digit number that indicates the server's result of the client's request. The description following the status code is just human-readable text that describes the status code. For example, this status line:

```
HTTP/1.0 200 OK
```

Indicates that the server uses version 1.0 of HTTP in its response. A status code of 200 means that the client's request was successful and the requested data will be supplied after the headers.

2. After the status line, the server sends header information to the client about itself and the requested document. For example:

   Date: Wed, 27 February 2013 01:17:58 GMT

   Server: CSA/2.0

   Last-modified: Wed, 20 February 2013 01:07:54 GMT

   Content-type: text/html

   Content-length: 2482

   A blank line ends the header.

3. If the client's request is successful, the requested data is sent. This data may be a copy of a file, or the response from a CGI program. If the client's request could not be fulfilled, additional data may be a human-readable explanation of why the server could not fulfill the request.

   In HTTP 1.0, after the server has finished sending the requested data, it disconnects from the client and the transaction is over unless a *Connection: Key Admin* header is sent. In HTTP 1.1, however, the default is for the server to maintain the connection and allow the client to make additional requests.

   Being a stateless protocol, HTTP does not maintain any information from one transaction to the next, so the next transaction needs to start all over again. The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there's no additional overhead for tracking sessions from one connection to the next.

SCRIPTING

Client-side scripting is the class of computer programs on the web that are executed in *client-side*, by the user's web browser, instead of *server-side* (on the web server). 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. Web authors write client-side scripts in languages such as JavaScript (Client-side JavaScript) and VBScript.

Client-side scripts are embedded within an HTML document (hence known as an "embedded script"), but they may also be contained in a separate file, which is referenced by the document (or documents) that uses it (known as an "external script"). Upon request, the necessary files are sent to the user's computer by the web server (or servers) on whom they reside. The user's web browser executes the script, and then displays the

Chapter 2

Server Side Scripting (PHP)

INTRODUCTION

Static vs. Dynamic websites

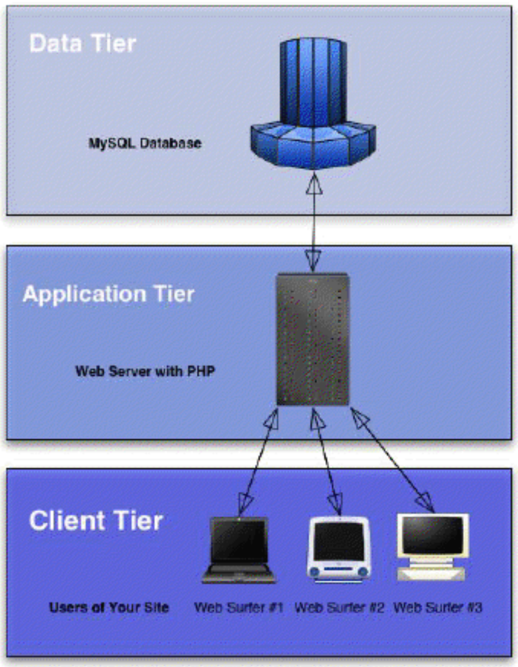
A static website is created using HTML pages which have been written in a fixed way to instruct the browser how to render the page. (What You See Is What You Get). Static websites are used to display fixed information that does not need to be updated on a regular basis. As the Web evolved, sites became more complex and were expected to display dynamic information that could not have been "hard coded" in HTML.

Dynamic websites

The solution was to generate the HTML "on-the-fly" based on parameters and specifications. For example, if a user asks to see search results, a unique page would be generated specifically for him. Since there is no way of knowing every possible search result for every possible keyword, the final page is being assembled using a server-side script. The server-side script constructs the page and then it is being sent back to the client. This means that the page result.html will look different each time based on the given specifications.

Every client makes a request for a page. The server checks what the client has asked for and based on that, it constructs a page for him on an existing template. The template is a page, which contains HTML code and server-side code that will end up as HTML code as well, but a different HTML each time.

Three Tier Web Applications



Benefits Of Dynamic Content

- The ability to customize pages for any given user.
- Greater ease of modification and update.
- Extended functionality (login, transactions, communication etc.).
- Automating the site and allowing it to support itself. Fewer pages to be created manually.

PHP Parser Installation:

Before you proceed it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP. Type the following address into your browser's address box.

```
http://127.0.0.1 or http://localhost/
```

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

Server-Side Scripting

In order to generate dynamic pages we use a server-side scripting language. There are different types of server-side scripting languages such as PHP, ASP, ASP.NET, ColdFusion, JavaServer Pages, Perl and others. Each scripting languages is being interpreted by an application, just as Flash is used to make sense of ActionScript and have the code do things. The application which interprets the server-side script is installed on the sever just like any other application.

Server-side scripting languages are also operating systems dependent. PHP for example, is being interpreted by an application called Apache. Each server-side scripting language supports basic programming concepts such as variables, arrays, functions, loops, conditional statement and others. They also contain more specific elements such as special objects, commands used to communicate with the server and a database and much more.

Server-Side Scripting & Databases

When there is a need to store and retrieve information (members, items in stock etc.) a database will be used to contain the data. Sever-side script can communicate to a database using a structured query language(SQL) which manipulates the database (add, remove, update, delete etc.)

What is 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 stands 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.
- PHP is an open source software (OSS), This means it's free to use and isn't being controlled by a single entity.
- It is being developed by a group of developers.
- PHP syntax resembles that of JavaScript and ActionScript in different ways.
- PHP is free to download and use
- PHP files may contain text, HTML tags and scripts.
- PHP files are returned to the browser as plain HTML.
- PHP files have a file extension of ".php", ".php3", or ".phtml".
- PHP can be written in any text editor.
- PHP script will be located inside special tags, much like JavaScript
  e.g. <?php //php script here ?>
- PHP code can be located any where in the page.
- PHP is case sensitive.
- Every variable in PHP will have the $ symbol as a prefix
  e.g. $myCollege ="KCC";
- Every line of code MUST be terminated with a semicolon ';'.

<u>Common uses of PHP:</u>

- 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, and 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.

What is MySQL?

- MySQL is a database server.
- MySQL is ideal for small and medium applications.
- MySQL supports standard SQL.
- MySQL compiles on a number of platforms.
- MySQL is free to download and use.

PHP + MySQL

PHP combined with MySQL is cross-platform (Means that you can develop in Windows and serve on a Linux platform as well).

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side.
- Reduce the time to create large websites.
- Create a customized user experience for visitors based on information that you have gathered from them.
- Open up thousands of possibilities for online tools like shopping carts for e-commerce websites.

What you should Know?

- HTML - Know the syntax and especially HTML Forms.
- Basic programming knowledge - This isn't required, but if you have any traditional programming experience it will make learning PHP a great deal easier.

<?PHP ... ?>

Syntax, Variables & Strings

Basic PHP Syntax

You cannot view the PHP source code by selecting "View source" in the browser – you will only see the output from the PHP file, which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser. A PHP scripting block always starts with <?php and ends with ?>. A PHP scripting block can be placed anywhere in the document. On servers with shorthand support enabled you can start a scripting block with <? and end with ?>. However, for maximum compatibility, recommended is that you use the standard form (<?php) rather than the shorthand form.

```
<?php ... ?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

```
<html>
<body>
<?php
        echo "Hello World";
?>
</body>
</html>
```

Each code line in PHP must end with a **semicolon**. The semicolon is a separator and is used to distinguish one set of instructions from another. There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

Comments In PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>
<?php
```

```
            //This is a comment

            /*
            This is a comment block.
            */
        ?>
        </body>
        </html>
```

PHP Variables
Variables are used for storing a value, like text strings, numbers or arrays. When a variable is set it can be used over and over again in your script. All variables in PHP start with a $ sign symbol. The correct way of setting a variable in PHP:

```
        $var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.
Let's try creating a variable with a string, and a variable with a number:

```
<?php
        $txt = "Hello World!";
        $number = 16;
?>
```

PHP is a Loosely Typed Language
In PHP a variable does not need to be declared before being set. In the previous example, you see that you do not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on how they are set. In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it (Like in C, C++ Programming.). In PHP the variable is declared automatically when you use it.

Variable Naming Rules

- A variable name must start with a letter or an underscore "_".
- A variable name can only contain alpha-numeric characters and underscores (a-Z, 0-9, and _).
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString).

<u>PHP Variable Scope</u>

The scope of a variable is the portion of the script in which the variable can be referenced.

PHP has four different variable scopes:

- local
- global
- static
- parameter

Local Scope

A variable declared **within** a PHP function is local and can only be accessed within that function. (the variable has local scope):

```
<?php
$a = 5; // global scope

function myTest()
{
echo $a; // local scope
}

myTest();
?>
```

The script above will not produce any output because the echo statement refers to the local scope variable $a, which has not been assigned a value within this scope. You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared. Local variables are deleted as soon as the function is completed.

Global Scope

Global scope refers to any variable that is defined outside of any function. Global variables can be accessed from any part of the script that is not inside a function. To access a global variable from within a function, use the **global** keyword:

```
<?php
$a = 5;
$b = 10;

function myTest()
{
global $a, $b;
$b = $a + $b;
}

myTest();
echo $b;
?>
```

The script above will output 15.

PHP also stores all global variables in an array called $GLOBALS[*index*]. Its index is the name of the variable. This array is also accessible from within functions and can be used to update global variables directly. The example above can be rewritten as this:

```
<?php
$a = 5;
$b = 10;

function myTest()
{
$GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

myTest();
echo $b;
?>
```

Static Scope

When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted.

To do this, use the **static** keyword when you first declare the variable:

```
static $rememberMe;
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

Parameters

A parameter is a local variable whose value is passed to the function by the calling code. Parameters are declared in a parameter list as part of the function declaration:

```
function myTest($para1,$para2,...)
{
    // function code
}
```

Parameters are also called arguments.

Strings In PHP

- String variables are used for values that contain character strings.

- After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the string "Hello World" to a string variable called $txt:

```
<?php
	$txt="Hello World";
	echo $txt;
?>
```

The output of the code will be:

```
	Hello World
```

PHP - string creation single quotes

Thus far we have created strings using double-quotes, but it is just as correct to create a string using single-quotes, otherwise known as apostrophes.

PHP Code:

```
$my_string = 'KCC - Unlock your potential!'; echo 'KCC - Unlock your potential!'; echo $my_string;
```

If you want to use a single-quote within the string you have to *escape* the single-quote with a backslash \ . Like this: \' !

PHP Code:

```
echo 'It\'s Neat!';
```

PHP - string creation double-quotes

We have used double-quotes and will continue to use them as the primary method for forming strings. Double-quotes allow for many special escaped characters to be used that you cannot do with a single-quote string. Once again, a backslash is used to escape a character.

PHP Code:

```
$newline = "A newline is \n"; $return = "A carriage return is \r"; $tab = "A tab is \t"; $dollar = "A dollar sign is \$"; $doublequote = "A double-quote is \"";
```

Note: If you try to escape a character that doesn't need to be, such as an apostrophe, then the backslash will show up when you output the string.

These escaped characters are not very useful for outputting to a web page because HTML ignore extra white space. A tab, newline, and carriage return are all examples of extra (ignorable) white space. However, when writing to a file that may be read by human eyes these escaped characters are a valuable tool!

PHP - string creation heredoc

The two methods above are the traditional way to create strings in most programming languages. PHP introduces a more robust string creation tool called *heredoc* that lets the programmer create multi-line strings without using quotations. However, creating a string using heredoc is more difficult and can lead to problems if you do not properly code your string! Here's how to do it:

PHP Code:

```
$my_string = <<<TEST
'KCC - Unlock your potential
TEST;
echo $my_string;
```

There are a few **very** important things to remember when using heredoc.

- Use <<< and some identifier that you choose to begin the heredoc. In this example we chose *TEST* as our identifier.
- Repeat the identifier followed by a semicolon to end the heredoc string creation. In this example that was *TEST;*
- The closing sequence *TEST;* must occur on a line by itself and **cannot** be indented!

Another thing to note is that when you output this multi-line string to a web page, it will not span multiple lines because we did not have any <br /> tags contained inside our string! Here is the output made from the code above.

Output:

```
'KCC - Unlock your potential
```

The Concatenation Operator

There is only one string operator in PHP. The concatenation operator (.) is used to put two string values together.To concatenate two variables together, use the dot (.) operator:

```
<?php
        $txt1="Hello World";
        $txt2="1234";
        echo $txt1 . " " . $txt2;
?>
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string. Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

The output of the code will be:

```
        Hello World 1234
```

Using the strlen() function

The strlen() function is used to find the length of a string. Let's find the length of our string "Hello world!"

```
<?php
        echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
        12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

- The strpos() function is used to search for a string or character within a string.
- If a match is found in the string, this function will return the position of the first match.
- If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
        echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

```
        6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Operators

In all programming languages, operators are used to manipulate or perform operations on variables and values. We have already seen the string concatenation operator "." and the assignment operator "=" in pretty much every PHP example so far.

There are many operators used in PHP, so we have separated them into the following categories to make it easier to learn them all.

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- String Operators
- Combination Arithmetic & Assignment Operators

The assignment operator = is used to assign values to variables in PHP. The arithmetic operator + is used to add values together.

Arithmetic Operators

The table below lists the arithmetic operators in PHP:

Operator	Name	Description	Example	Result
x + y	Addition	Sum of x and y	2 + 2	4
x - y	Subtraction	Difference of x and y	5 - 2	3
x * y	Multiplication	Product of x and y	5 * 2	10
x / y	Division	Quotient of x and y	15 / 5	3
x % y	Modulus	Remainder of x divided by y	5 % 2 10 % 8 10 % 2	1 2 0
- x	Negation	Opposite of x	- 2	
a . b	Concatenation	Concatenate two strings	"Hi" . "Ha"	HiHa

Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "$x = 5" is 5.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus
a .= b	a = a . b	Concatenate two strings

Incrementing/Decrementing Operators

Operator	Name	Description
++ x	Pre-increment	Increments x by one, then returns x
x ++	Post-increment	Returns x, then increments x by one
-- x	Pre-decrement	Decrements x by one, then returns x
x --	Post-decrement	Returns x, then decrements x by one

Comparison Operators

Comparison operators allows you to compare two values:

Operator	Name	Description	Example
x == y	Equal	True if x is equal to y	5==8 returns false
x === y	Identical	True if x is equal to y, and they are of same type	5==="5" returns false
x != y	Not equal	True if x is not equal to y	5!=8 returns true
x <> y	Not equal	True if x is not equal to y	5<>8 returns true
x !== y	Not identical	True if x is not equal to y, or they are not of same type	5!=="5" returns true
x > y	Greater than	True if x is greater than y	5>8 returns false
x < y	Less than	True if x is less than y	5<8 returns true
x >= y	Greater than or equal to	True if x is greater than or equal to y	5>=8 returns false
x <= y	Less than or equal to	True if x is less than or equal to y	5<=8 returns true

Logical Operators

Operator	Name	Description	Example

x and y	And	True if both x and y are true	x=6
			y=3
			(x < 10 and y > 1) returns true

x or y	Or	True if either or both x and y are true	x=6
			y=3
			(x==6 or y==5) returns true

x xor y	Xor	True if either x or y is true, but not both	x=6
			y=3
			(x==6 xor y==3) returns false

x && y	And	True if both x and y are true	x=6
			y=3
			(x < 10 && y > 1) returns true

x || y	Or	True if either or both x and y are true	x=6
			y=3
			(x==5 || y==5) returns false

! x	Not	True if x is not true	x=6
			y=3
			!(x==y) returns true

Array Operators

Operator	Name	Description
x + y	Union	Union of x and y
x == y	Equality	True if x and y have the same key/value pairs
x === y	Identity	True if x and y have the same key/value pairs in the same order and of the same types
x != y	Inequality	True if x is not equal to y
x <> y	Inequality	True if x is not equal to y
x !== y	Non-identity	True if x is not identical to y

Conditional Statements

In PHP 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 a condition is true and another code if the condition is false
- **if...elseif....else statement -** use this statement to select one of several blocks of code to be executed
- **switch statement -** use this statement to select one of many blocks of code to be executed

---

The 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;*

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

Note that there is no ‘else’ in this syntax. The code is executed **only if the specified condition is true**.

The if...else Statement

Use the if... else statement to execute some code if a condition is true and another code if a condition is false.

Syntax

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

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Have a nice weekend!";
  }
else
  {
  echo "Have a nice day!";
  }
?>

</body>
</html>
```

The if... elseif... else Statement

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

Syntax

```
if (condition)
 {
 code to be executed if condition is true;
 }
elseif (condition)
 {
 code to be executed if condition is true;
 }
else
 {
 code to be executed if condition is false;
 }
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
 {
 echo "Have a nice weekend!";
 }
elseif ($d=="Sun")
 {
 echo "Have a nice Sunday!";
 }
else
 {
 echo "Have a nice day!";
 }
?>

</body>
</html>
```

The PHP Switch Statement

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

Syntax

```
switch (n)
{
case label1:
  code to be executed if n=label1;
  break;
case label2:
  code to be executed if n=label2;
  break;
default:
  code to be executed if n is different from both label1 and label2;
}
```

This is how it works:

First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<html>
<body>

<?php
$x=1;
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

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

PHP 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 PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

---

The while Loop

The while loop executes a block of code while a condition is true.

Syntax

```
while (condition)
 {
 code to be executed;
 }
```

Example

The example below first sets a variable *i* to 1 ($i=1;).

Then, the while 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:

```
<html>
<body>

<?php
$i=1;
while($i<=5)
 {
 echo "The number is " . $i . "<br>";
```

```
  $i++;
  }
?>

</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

Syntax

```
do
 {
 code to be executed;
 }
while (condition);
```

Example

The example below first sets a variable *i* to 1 ($i=1;).

Then, it starts the do...while loop. The loop will increment the variable *i* with 1, and then write some output. Then the condition is checked (is *i* less than, or equal to 5), and the loop will continue to run as long as *i* is less than, or equal to 5:

```
<html>
<body>

<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br>";
```

```
 }
while ($i<=5);
?>

</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

<u>The for Loop</u>

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

Syntax

```
for (init; condition; increment)
  {
  code to be executed;
  }
```

Parameters:

- *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the iteration)

Note: The *init* and *increment* parameters above can be empty or have multiple expressions (separated by commas).

Example

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

```
<html>
<body>

<?php
```

```
for ($i=1; $i<=5; $i++)
 {
 echo "The number is " . $i . "<br>";
 }
?>

</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The foreach Loop

The foreach loop is used to loop through arrays.

Syntax

```
foreach ($array as $value)
 {
 code to be executed;
 }
```

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
 {
 echo $value . "<br>";
 }
?>
```

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

Output:

```
one
two
three
```

An array stores multiple values in one single variable.

Array

Like arrays in other languages, PHP arrays allow you to store multiple values in a single variable and operate on them as a set.

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value. An array is a special variable, which can store multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array (*Enumerated Arrays*) -** An array with a numeric index
- **Associative array -** An array where each ID key is associated with a value
- **Multidimensional array -** An array containing one or more arrays

---

Numeric Arrays (Enumerated Arrays)

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";

echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
  (
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
  );
```

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] . " a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

Array functions

PHP offers an extensive array manipulation toolkit—over 60 functions—that lets you process arrays in almost any way imaginable including reversing them, extracting subsets, comparing and sorting, recursively processing, and searching them for specific values.

Function	**Explanation**	**Example**
sizeof($arr)	This function returns the number of elements in an array. Use this function to find out how many elements an array contains; this information is most commonly used to initialize a loop counter when processing the array.	Code: $data = array("red", "green", "blue"); echo "Array has " . sizeof($data) . " elements"; ?> Output: Array has 3 elements
array_values($arr)	This function accepts a PHP array and returns a new array containing only its values (not its keys). Its counterpart is the array_keys() function. Use this function to retrieve all the values from an associative array.	Code: $data = array("hero" => "Holmes", "villain" => "Moriarty"); print_r(array_values($data)); ?> Output: Array ([0] => Holmes [1] => Moriarty)
array_keys($arr)	This function accepts a PHP array and returns a new array containing only its keys (not its values). Its counterpart is the array_values() function. Use this function to retrieve all the keys from an associative array.	Code: $data = array("hero" => "Holmes", "villain" => "Moriarty"); print_r(array_keys($data)); ?> Output: Array ([0] => hero [1] => villain)
array_pop($arr)	This function removes an element from the end of an	Code: $data = array("Donald", "Jim", "Tom");

	array.	array_pop($data); print_r($data); ?> Output: Array ([0] => Donald [1] => Jim)
array_push($arr, $val)	This function adds an element to the end of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_push($data, "Harry"); print_r($data); ?> Output: Array ([0] => Donald [1] => Jim [2] => Tom [3] => Harry)
array_shift($arr)	This function removes an element from the beginning of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_shift($data); print_r($data); ?> Output: Array ([0] => Jim [1] => Tom)
array_unshift($arr, $val)	This function adds an element to the beginning of an array.	Code: $data = array("Donald", "Jim", "Tom"); array_unshift($data, "Sarah"); print_r($data); ?> Output: Array ([0] => Sarah [1] => Donald

		[2] => Jim [3] => Tom)

each($arr)	This function is most often used to iteratively traverse an array. Each time each() is called, it returns the current key-value pair and moves the array cursor forward one element. This makes it most suitable for use in a loop.	Code:
		$data = array("hero" => "Holmes", "villain" => "Moriarty");
		while (list($key, $value) = each($data)) {
		echo "$key: $value \n";
		}
		?>
		
		Output:
		hero: Holmes
		villain: Moriarty

sort($arr)	This function sorts the elements of an array in ascending order. String values will be arranged in ascending alphabetical order.
	
	Note: Other sorting functions include asort(), arsort(), ksort(), krsort() and rsort().
		Code:
		$data = array("g", "t", "a", "s");
		sort($data);
		print_r($data);
		?>
		
		Output:
		Array
		(
		[0] => a
		[1] => g
		[2] => s
		[3] => t
		)

array_flip($arr)	The function exchanges the keys and values of a PHP associative array. Use this function if you have a tabular (rows and columns) structure in an array, and you want to interchange the rows and columns.	Code: $data = array("a" => "apple", "b" => "ball"); print_r(array_flip($data)); ?> Output: Array ([apple] => a [ball] => b)
array_reverse($arr)	The function reverses the order of elements in an array. Use this function to re-order a sorted list of values in reverse	Code: $data = array(10, 20, 25, 60); print_r(array_reverse($data)); ?> Output: Array

	for easier processing—for example, when you're trying to begin with the minimum or maximum of a set of ordered values.	([0] => 60 [1] => 25 [2] => 20 [3] => 10)
array_merge($arr)	This function merges two or more arrays to create a single composite array. Key collisions are resolved in favor of the latest entry. Use this function when you need to combine data from two or more arrays into a single structure—for example, records from two different SQL queries.	Code: $data1 = array("cat", "goat"); $data2 = array("dog", "cow"); print_r(array_merge($data1, $data2)); ?> Output: Array ([0] => cat [1] => goat [2] => dog [3] => cow)
array_rand($arr)	This function selects one or more random elements from an array. Use this function when you need to randomly select from a collection of discrete values—for example, picking a random color from a list.	Code: $data = array("white", "black", "red"); echo "Today's color is " . $data[array_rand($data)]; ?> Output: Today's color is red
array_search($search, $arr)	This function searches the values in an array for a match to the search term, and returns the corresponding key if found. If more than one match exists, the key of the first matching value is	Code: $data = array("blue" => "#0000cc", "black" => "#000000", "green" => "#00ff00"); echo "Found " . array_search("#0000cc", $data); ?> Output: Found blue

	returned. Use this function to scan a set of index-value pairs for matches, and return the matching index.	
array_slice($arr, $offset, $length)	This function is useful to extract a subset of the elements of an array, as another array. Extracting begins from array offset $offset and continues until the array slice is $length elements long. Use this function to break a larger array into smaller ones—for example, when segmenting an array by size ("chunking") or type of data.	Code: $data = array("vanilla", "strawberry", "mango", "peaches"); print_r(array_slice($data, 1, 2)); ?> Output: Array ([0] => strawberry [1] => mango)
array_unique($data)	This function strips an array of duplicate values. Use this function when you need to remove non-unique elements from an array—for example, when creating an array to hold values for a table's primary key.	Code: $data = array(1,1,4,6,7,4); print_r(array_unique($data)); ?> Output: Array ([0] => 1 [3] => 6 [4] => 7 [5] => 4)

array_walk($arr, $func)	This function "walks" through an array, applying a user-defined function to every element. It returns the changed array.
	
	Use this function if you need to perform
		Code:
		function reduceBy10(&$val, $key) {
		$val -= $val * 0.1;
		}
		
		$data = array(10,20,30,40);
		array_walk($data, 'reduceBy10');
		print_r($data);
		?>

	custom processing on every element of an array—for example, reducing a number series by 10%.	Output: Array ([0] => 9 [1] => 18 [2] => 27 [3] => 36)

PHP Functions

A function is just a name we give to a block of code that can be executed whenever we need it. This might not seem like that big of an idea, but believe me, when you understand and use functions you will be able to save a ton of time and write code that is much more readable!

For example, you might have a company motto that you have to display at least once on every webpage. If you don't, then you get fired! Well, being the savvy PHP programmer you are, you think to yourself, "this sounds like a situation where I might need functions."

Tip: Although functions are often thought of as an advanced topic for beginning programmers to learn, if you take it slow and stick with it, functions can be just minor speed bump in your programming career. So don't give up if functions confuse you at first!

To keep the script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

+ PHP Built-in Functions

- PHP Array Functions
- PHP Calendar Functions
- PHP Date & Time Functions
- PHP Error Handling Functions
- PHP File System Functions
- PHP MySQL Functions
- PHP String Functions

+ PHP User Defined Functions

Create a PHP Function

A function will be executed by a call to the function.

Syntax

```
function functionName()
{
code to be executed;
}
```

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes
```

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>

<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br>";
}

echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br>";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

PHP Functions - Return values

To let a function return a value, use the return statement.

Example

```
<html>
<body>

<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

Output:

```
1 + 16 = 17
```

PHP String Functions

substr()

This function returns the part of the string as an output.

Syntax :

substr(<string>,<start>,[<length>]);

Explanation :

String : It is mandatory parameter. The string from which the part is to be extracted is mentioned here.

Start : The start in the string from which the characters are to be extracted

- Positive number – Start at a specified position in the string
- Negative number – Start at a specified position from the end of the string
- 0 – Start at the first character in string

Length : It is an optional parameter. It specifies the length of the string which is to be extracted.

- Positive number – The length to be returned from the start parameter
- Negative number – The length to be returned from the end of the string

Example 1:

<?php echo substr("Hello world",6); ?> //Returns world

Example 2 :

<?php echo substr("Hello world",6,4); ?> // Returns worl

Example 3 :

<?php echo substr("Hello world", -1); ?> // Returns d

Example 4:

<?php echo substr("Hello world", -3, -1); ?> // Returns rl

strlen()

This function returns the length of the string

Syntax :

strlen(<string>);

Explanation:

String : It is mandatory field. The string whose length is to be found out is mentioned here.

Example 1:

<?php echo strlen("Hello world"); ?> // Returns 11

trim()

This function removes the whitespaces from both start and the end of the string.

Syntax :

trim(<string>);

Explanation :

String : It is mandatory field. The string of which the whitespaces are to be removed is passed as parameter.

Example 1:

```
<?php echo trim( "       Hello World    "); ?>
```

Returns Hello World. If you go view source then you can see that there are no whitespaces.

ltrim()

This function removes the whitespaces from the left part of the string.

Syntax :

ltrim(<string>);

Explanation :

String : It is mandatory field. The string of which the whitespaces are to be removed from left side is passed as parameter.

Example 1:

```
<?php echo ltrim( "       Hello World    "); ?>
```

Returns Hello World. If you go view source then you can see that there are no whitespaces on left side but there are spaces on right side.

rtrim()

This function removes the whitespaces from the right part of the string.

Syntax :

rtrim(<string>);

Explanation :

String : It is mandatory field. The string of which the whitespaces are to be removed from right side is passed as parameter.

Example 1:

```
<?php echo rtrim( "       Hello World    "); ?>
```

Returns Hello World. If you go view source then you can see that there are no whitespaces on right side but there are spaces on left side.

strtolower()

This function converts the string to lower case

Syntax :

```
strtolower(<string>);
```

Explanation :

String : It is mandatory field. The string which is to be converted to lower case is passed here.

Example 1:

```
<?php echo strtolower("HELLO WORLD"); ?> Returns hello world
```

strtoupper()

This function converts the string to upper case

Syntax :

```
strtoupper(<string>);
```

Explanation :

String : It is mandatory field. The string which is to be converted to upper case is passed here.

Example 1:

```
<?php echo strtoupper("hello world"); ?> // Returns HELLO WORLD
```

str_replace()

The str_replace() function replaces some characters with some other characters in a string.
This function works by the following rules:

- If the string to be searched is an array, it returns an array
- If the string to be searched is an array, find and replace is performed with every array element
- If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace
- If find is an array and replace is a string, the replace string will be used for every find value

Syntax :

```
str_replace(<search>,<replace>,<string/array>,[<count>]);
```

Explanation :

Search : It is mandatory . The string or value to be searched comes here.

Replace : It is mandatory. The string or value to be replaced comes here.

String/Array : It is mandatory. The string or array in which the value is to be found out comes here.

Count : It is optional. It counts the number of replacements to be done.

Example 1:

```
<?php echo str_replace("world","Peter","Hello world"); ?>// Returns Hello Peter
```

Example 2:

```
<?php
 $arr = array("blue","red","green","yellow");
 print_r(str_replace("red","pink",$arr,$i));
 echo "Replacements: $i";
 ?>
```

Output:

```
Array
 (
 [0] => blue
 [1] => pink
 [2] => green
 [3] => yellow
 )
 Replacements: 1
```

strcmp()

The strcmp() function compares two strings.

This function returns:

- 0 – if the two strings are equal
- <0 – if string1 is less than string2
- >0 – if string1 is greater than string2

Syntax :

```
strcmp(<string1>,<string2>);
```

Explanation :

String1 : It is mandatory. The first string comes here.

String 2 : It is mandatory. The Second string comes here.

Example 1:

```
<?php echo strcmp("Hello world!","Hello world!"); ?>

//Returns 0
```

Note: The strcmp() function is binary safe and case-sensitive. For case insensitive comparison you can use strcasecmp(<string1>,<string2>); function. It is similar to strcmp() function.

explode()

This function breaks the string into array on the basis of delimiter passed.

Syntax:

```
explode(<delimeter>,<string>,[<limit>]);
```

Explanation:

Delimeter: It is mandatory field. It specifies where to break the string.

String: It is mandatory. It specifies the string to split.

Limit : It is optional. It specifies the maximum number of array elements to return.

Example 1:

```
<?php
 $str = "Hello world. It's a beautiful day.";
 print_r (explode(" ",$str));
 ?>
```

Output :

```
Array
 (
 [0] => Hello
 [1] => world.
 [2] => It's
 [3] => a
 [4] => beautiful
 [5] => day.
 )
```

implode()

This function join array elements with a string on the basis of delimiter passed.

Syntax:

```
implode(<delim>,<array>);
```

Explanation:

Delimiter: It is mandatory field. It specifies what to put between the array elements. Default is “” (an empty string).

Array: It is mandatory field. It specifies the array to join to a string.

Example 1:

```
<?php
 $arr = array('Hello','World!','Beautiful','Day!');
 echo implode(" ",$arr);
 ?>
```

Output:

```
Hello World! Beautiful Day!
```

File Handling

Opening a File

The fopen() function is used to open files in PHP. The first parameter of this function contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened:

```
<html>
<body>

<?php
        $file=fopen("welcome.txt","r");
?>

</body>
</html>
```

The file may be opened in one of the following modes:

Modes	Description
r	Read only. Starts at the beginning of the file
r+	Read/Write. Starts at the beginning of the file
w	Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist
w+	Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist
a	Append. Opens and writes to the end of the file or creates a new file if it doesn't exist
a+	Read/Append. Preserves file content by writing to the end of the file
x	Write only. Creates a new file. Returns FALSE and an error if file already exists
x+	Read/Write. Creates a new file. Returns FALSE and an error if file already exists

Note: If the fopen() function is unable to open the specified file, it returns 0 (false).

Example

The following example generates a message if the fopen() function is unable to open the specified file:

```
<html>
<body>

<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
```

```
?>

</body>
</html>
```

Closing a File

The fclose() function is used to close an open file:

```
<?php
$file = fopen("test.txt","r");

//some code to be executed

fclose($file);
?>
```

Check End-of-file

The feof() function checks if the "end-of-file" (EOF) has been reached. The feof() function is useful for looping through data of unknown length.

Note: You cannot read from files opened in w, a, and x mode!

```
if (feof($file)) echo "End of file";
```

Reading a File Line by Line

The fgets() function is used to read a single line from a file.

Note: After a call to this function the file pointer has moved to the next line.

Example

The example below reads a file line by line, until the end of file is reached:

```
<?php
$file = fopen("welcome.txt", "r") or exit("Unable to open file!");
//Output a line of the file until the end is reached
while(!feof($file))
  {
  echo fgets($file). "<br>";
  }
```

```
fclose($file);
?>
```

Reading a File Character by Character

The fgetc() function is used to read a single character from a file.

Note: After a call to this function the file pointer moves to the next character.

Example

The example below reads a file character by character, until the end of file is reached:

```
<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
while (!feof($file))
  {
  echo fgetc($file);
  }
fclose($file);
?>
```

DATABASE PROGRAMMING WITH PHP

PHP has support for over 20 databases, including the most popular commercial and open source varieties. Relational database systems such as MySQL, PostgreSQL, and Oracle are the backbone of most modern dynamic web sites. In these are stored shopping-cart information, purchase histories, product reviews, user information, credit-card numbers, and sometimes-even web pages themselves.

SQL

SQL stands for Structured Query Language, and it is a very powerful and diverse language used to create and query databases.

The loose structure and flexibility of this language make it an ideal candidate for the web. There are many database applications available for developers to use for free, such as MySQL.

Some of the basic functions of SQL are inputting, modifying, and dropping data from databases. Coupled with the use of web languages such as HTML and PHP, SQL becomes an even greater tool for building dynamic web applications.

With a SQL backend, it is fairly simple to store user data, email lists, or other kinds of dynamic data. E-Commerce web sites, community sites, and online web services rely on SQL databases to manage user data or process user purchases. SQL has become popular among web developers due to its flexibility and simplicity. With some basic knowledge of HTML, PHP, and a database program such as MySQL, a developer becomes capable of creating complex websites and applications while relying on online web services to provide a SQL backend in which user data is stored.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and view

DML and DDL

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The query and update commands form the DML part of SQL:

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database

The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specifies links between tables, and imposes constraints between tables. The most important DDL statements in SQL are:

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

What's a Database?

A SQL database is nothing more than an empty shell, like a vacant warehouse. It offers no real functionality whatsoever, but does provide a virtual space to store data. Data is stored inside of database objects called tables, and tables are the containers that actually hold specific types of data, such as numbers, files, strings, and dates. A single database can house hundreds of tables containing 1,000s of table columns each.

SQL coins the term query as the name for its commands. Basically, all SQL code is written in the form of a query statement and then executed against a database. All SQL queries perform some type of data operation such as selecting data, inserting/updating data, or creating data objects such as SQL databases and SQL tables. Each query statement begins with a clause such as SELECT, UPDATE, CREATE or DELETE.

CREATE

SQL CREATE is the command used to create data objects, including everything from new databases and tables to views and stored procedures.

SQL Create Database Query:

```
CREATE DATABASE MyDatabase;
```

SQL is not case sensitive.

Data is stored inside SQL tables, which are contained within SQL databases. A single database can house hundreds of tables, each playing its own unique role in the database schema.

SQL tables are comprised of table rows and columns. Table columns are responsible for storing many different types of data, like numbers, texts, dates, and even files. There are many different types of table columns and these data types vary, depending on how the SQL table has been created by the SQL developer. A table row is a horizontal record of values that fit into each different table column.

SQL Create Table Query:

```
USE mydatabase;
CREATE TABLE orders (
        id INT IDENTITY(1,1) PRIMARY KEY,
        customer VARCHAR(50),
        day_of_order DATETIME,
        product VARCHAR(50),
        quantity INT);
```

The first line of the example, "USE mydatabase;", is pretty straightforward. This line defines the query scope and directs SQL to run the command against the MyDatabase object we created earlier.

The line starting with the CREATE clause is where we are actually going to tell SQL to create the new table, which is named orders. Each table column has its own set of guidelines or schema, and the lines of code above contained in parenthesis () are telling SQL how to go about setting up each column schema. Table columns are presented in list format, and each schema is separated with a comma (,).

INSERT

INSERT command will insert a new data row into SQL table, orders.

```
INSERT INTO orders (customer,day_of_order,product, quantity)
VALUES('KCC','8/1/08','Pen',4);
```

SELECT

SELECT queries are the most commonly used SQL commands, SELECT query will return records from the orders table that was created previously.

SQL Select Query Template:

SELECT table_column1, table_column2, table_column3 FROM my_table;

e.g. SELECT * FROM orders; {* refers to all.}

Select queries require two essential parts. The first part is the "WHAT", which determines what we want SQL to go and fetch. The second part of any SELECT command is the "FROM WHERE". It identifies where to fetch the data from, which may be from a SQL table, a SQL view, or some other SQL data object.

e.g. SELECT customer, day_of_order, product, quantity FROM orders;

WHERE CONDITION

The WHERE clause sets a conditional statement, and it can be used with any type of SQL query. As the select query executes, SQL processes one row at a time. Each time the conditional statement is met (returns true), a row is returned as a result. SQL WHERE is essentially, a filtering mechanism for SQL queries and is a tremendous asset to any aspiring SQL developer.

e.g. SELECT * FROM orders WHERE customer = 'KCC'

UPDATE

SQL UPDATE is the command used to update existing table rows with new data values. UPDATE is a very powerful command in the SQL world. It has the ability to update every single row in a database with the execution of only a single query. Due to UPDATE's supreme authority, it is best to always include a WHERE clause when working with UPDATE query statements. That way, there won't be any chances that accidentally update more rows.

Syntax: UPDATE table_name SET values … WHERE condition;

UPDATE orders SET quantity = '18', Product = External Disk' WHERE id = '1'

DELETE

In the SQL world, databases, rows, and columns all have one thing in common: once a DELETE statement has been executed successfully against them, the data they once contained is lost forever!

DELETE - Deletes any number of rows from a data object. DELETE queries work much like UPDATE queries and like UPDATE, it is much advised to always use a WHERE condition when running any delete query or else you risk deleting too much data.

```
DELETE FROM orders WHERE customer = 'KCC’;
```

DROP - Removes table columns, tables, and all data objects SQL applications. SQL DROP is another command that removes data from the data store. The DROP command must be performed on SQL objects including databases, tables, table columns, and SQL views. Dropping any of these objects removes them completely from your SQL application and all data contained in any of the data objects dropped are lost forever.

```
DROP TABLE orders;
DROP DATABASE mydatabase;
DROP COLUMN column_name
```

TRUNCATE - SQL TRUNCATE is the fastest way to remove all data from a SQL table, leaving nothing but an empty shell. You might choose to use this command when all the data inside of a table needs to be removed but you'd like the table column definitions to remain intact.

```
TRUNCATE TABLE orders;
```

[Refer to the subject ‘ADVANCED DBMS for details on SQL.]

CRUD

In computer programming, **create, read, update and delete** (**CRUD**) are the four basic functions of persistent storage. Sometimes *CRUD* is expanded with the words *retrieve* instead of *read*, *modify* instead of *update* or *destroy* instead of *delete*. It is also sometimes used to describe user interface conventions that facilitate viewing, searching, and changing information; often using computer-based forms and reports.

Another variation of CRUD is BREAD, an acronym for "Browse, Read, Edit, Add, Delete".

The acronym CRUD refers to all of the major functions that are implemented in relational database applications. Each letter in the acronym can map to a standard SQL statement.

Operation	**SQL**
Create	CREATE, INSERT
Read (Retrieve)	SELECT
Update (Modify)	UPDATE
Delete (Destroy)	DELETE

Database Connectivity

MySQL is the most popular open-source database system. MySQL is a database. The data in MySQL is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows. Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

The free MySQL database is very often used with PHP.

Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database. In PHP, this is done with the mysql_connect() function.

Syntax

```
mysql_connect(servername,username,password);
```

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
```

```
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code
?>
```

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql_close() function:

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code

mysql_close($con);
?>
```

Insert Data Into a Table

```
<html>
<body>

<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname">
Lastname: <input type="text" name="lastname">
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php". The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables. Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table.

Here is the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";

if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";

mysql_close($con);
?>
```

Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

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

```
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysql_fetch_array($result))
 {
 echo "<tr>";
 echo "<td>" . $row['FirstName'] . "</td>";
 echo "<td>" . $row['LastName'] . "</td>";
 echo "</tr>";
 }
echo "</table>";

mysql_close($con);
?>
```

Update Data In a Database

Update SQL Syntax

```
UPDATE table_name
SET values (e.g. column1=value, column2=value2, ...)
WHERE some_column=some_value
```

The following example updates some data in the "Persons" table:

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("my_db", $con);

mysql_query("UPDATE Persons SET Age=36
WHERE FirstName='Ram' AND LastName='Karki'");

mysql_close($con);
?>
```

Delete Data

Syntax

```
DELETE FROM table_name
WHERE some_column = some_value
```

The following example deletes all the records in the "Persons" table where LastName='Karki':

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
mysql_query("DELETE FROM Persons WHERE LastName='Karki'");
mysql_close($con);
?>
```

Some commonly used MySQL functions are:

- mysql_connect - Open a connection to a MySQL Server
- mysql_select_db - Select a MySQL database
- mysql_close - Close MySQL connection
- mysql_query - Send a MySQL query to the current database
- mysql_affected_rows - Get number of affected rows in previous MySQL operation
- mysql_client_encoding - Returns the name of the character set
- mysql_errno - Returns the numerical value of the error message from previous MySQL operation
- mysql_error - Returns the text of the error message from previous MySQL operation
- mysql_fetch_array - Fetch a result row as an associative array, a numeric array, or both
- mysql_fetch_lengths - Get the length of each output in a result
- mysql_fetch_object - Fetch a result row as an object
- mysql_fetch_row - Get a result row as an enumerated array

- mysql_field_len - Returns the length of the specified field
- mysql_field_name - Get the name of the specified field in a result
- mysql_field_table - Get name of the table the specified field is in
- mysql_field_type - Get the type of the specified field in a result
- mysql_info - Get information about the most recent query
- mysql_insert_id - Get the ID generated from the previous INSERT operation
- mysql_num_fields - Get number of fields in result
- mysql_num_rows - Get number of rows in result
- mysql_field_seek - Set result pointer to a specified field offset
- mysql_fetch_field - Get column information from a result and return as an object
- mysql_result - Get result data

For Parameter, Return Type and Description, refer:

http://php.net/manual/en/ref.mysql.php

http://www.w3schools.com/php/php_ref_mysql.asp

PHP Database ODBC: 'BONUS'

ODBC is an Application Programming Interface (API) that allows you to connect to a data source (e.g. an MS Access database).

Create an ODBC Connection

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the **Administrative Tools** icon in your Control Panel.
2. Double-click on the **Data Sources (ODBC)** icon inside.
3. Choose the **System DSN** tab.
4. Click on **Add** in the System DSN tab.

5. Select the **Microsoft Access Driver**. Click **Finish.**
6. In the next screen, click **Select** to locate the database.
7. Give the database a **Data Source Name (DSN)**.
8. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to set up a DSN for you to use.

Connecting to an ODBC

The odbc_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type.

The odbc_exec() function is used to execute an SQL statement.

Example

The following example creates a connection to a DSN called northwind, with no username and no password. It then creates an SQL and executes it:

```
$conn=odbc_connect('northwind','','');
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

Retrieving Records

The odbc_fetch_row() function is used to return records from the result-set. This function returns true if it is able to return rows, otherwise false.

The function takes two parameters: the ODBC result identifier and an optional row number:

```
odbc_fetch_row($rs)
```

Retrieving Fields from a Record

The odbc_result() function is used to read fields from a record. This function takes two parameters: the ODBC result identifier and a field number or name.

The code line below returns the value of the first field from the record:

```
$compname=odbc_result($rs,1);
```

The code line below returns the value of a field called "CompanyName":

```
$compname=odbc_result($rs,"CompanyName");
```

Closing an ODBC Connection

The odbc_close() function is used to close an ODBC connection.

```
odbc_close($conn);
```

An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

```
<html>
<body>

<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
  {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

```
if (!$rs)
  {exit("Error in SQL");}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
  {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
  }
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

PHP include and require Statements

In PHP, we can insert the content of one PHP file into another PHP file before the server executes it. The **include** and **require** statements are used to insert useful codes written in other files, in the flow of execution.

Include and require are identical, except upon failure:

- **require** will produce a fatal error (E_COMPILE_ERROR) and stop the script
- **include** will only produce a warning (E_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use include. Otherwise, in case of FrameWork, CMS or a complex PHP application coding, always use require to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

include '*filename*';

or

require '*filename*';

PHP include and require Statement

Basic Example

Assume that you have a standard header file, called "header.php". To include the header file in a page, use include/require:

```
<html>
<body>
<?php include 'header.php'; ?>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

Example 2

Assume we have a standard menu file that should be used on all pages.

"menu.php"

File Content

```
echo '<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/about.php">About Us</a>
<a href="/contact.php">Contact Us</a>';
```

All pages in the Web site should include this menu file. Here is how it can be done:

```
<html>
<body>

<div class="leftmenu">
<?php include 'menu.php'; ?>
</div>
```

```
<h1>Welcome to my home page.</h1>
<p>Some text.</p>
</body>
</html>
```

Example 3

Assume we have an include file with some variables defined ("vars.php"):

```
<?php
$color='red';
$car='BMW';
?>
```

Then the variables can be used in the calling file:

```
<html>
<body>

<h1>Welcome to my home page.</h1>
<?php include 'vars.php';
echo "I have a $color $car"; // I have a red BMW
?>

</body>
</html>
```

SESSION

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the Internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Sessions are a combination of a server-side cookie and a client-side cookie, where the client-side cookie is simply a reference id to the information stored in the server-side cookie. Here are some features of sessions:

- Server-site cookie can store very large amounts of data while regular cookies are limited in size.
- Since the client-side cookie generated by a session only contains the id reference (a random string of 32 hexadecimal digits, such as 'fca17f071bbg9bf7f85ca281653499a4' called a 'session id') you save on bandwidth.
- Much more secure than regular cookies since the data is stored on the server and cannot be edited by the user.
- Only last until the user closes their browser.
- Won't work if client has cookies disabled in their browser unless some extra measures are taken.
- Can be easily customized to store the information created in the session to a database.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?php
        session_start();
?>

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

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
    session_start();
    // Store session data
   $_SESSION['views']=1;
?>

<html>
<body>
<?php
     //Retrieve session data
     echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

Output:

```
Pageviews=1
```

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

```
<?php
   session_start();
   if(isset($_SESSION['views']))
   $_SESSION['views']=$_SESSION['views']+1;
   else
   $_SESSION['views']=1;
   echo "Views=". $_SESSION['views'];
?>
```

Destroying a Session

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

The unset() function is used to free the specified session variable:

```
<?php
   session_start();
   if(isset($_SESSION['views']))
   unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

```
<?php
   session_destroy();
?>
```

> ***Note:*** *session_destroy() will reset your session and you will lose all your stored session data.*

COOKIES

<u>What is a Cookie?</u>

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser; it will send the cookie too. With PHP, you can both create and retrieve cookie values.

How to Create a Cookie?

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

```
setcookie(name, value, expire, path, domain);
```

Example 1

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>
<html>
```

Note: *The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).*

Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "Alex Porter", $expire);
?>
<html>
```

In the example above the expiration time is set to a month (*60 sec * 60 min * 24 hours * 30 days*).

How to Retrieve a Cookie Value?

The PHP $_COOKIE variable is used to retrieve a cookie value.
In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
// Print a cookie
echo $_COOKIE["user"];
// A way to view all cookies
print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html>
<body>
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br>";
else
  echo "Welcome guest!<br>";
?>
</body>
</html>
```

How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time()-3600);
?>
```

The Difference Between Cookies and Sessions

The main difference between cookies and sessions is that cookies are stored in the user's browser, and sessions are not. This difference determines what each is best used for.

A cookie can keep information in the user's browser until deleted. If a person has a login and password, this can be set as a cookie in their browser so they do not have to re-login to your website every time they visit. You can store almost anything in a browser cookie. The trouble is that a user can block cookies or delete them at any time. If, for example, your website's shopping cart utilized cookies, and a person had their browser set to block them, then they could not shop at your website.

Sessions are not reliant on the user allowing a cookie. They work instead like a token allowing access and passing information while the user has their browser open. The problem with sessions is that when you close your browser you also lose the session. So, if you had a site requiring a login, this couldn't be saved as a session like it could, as a cookie, and the user would be forced to re-login every time they visit.

Cookies

1. Value stored in cookies is in the form of string.
2. They are stored at Client side
3. Cookie is non-secure since stored in text format at client side
4. Cookies may or may not be individual for every client
5. Due to cookies network traffic will increase.
6. We can use cookies only in few situations because of no security.
7. We can disable cookies.
8. Since the value is string there is no security.
9. We have persistent and non-persistent cookies.

Session

1. Session can store any type of data because the value is of data type of "object".
2. These are stored at Server side.
3. Session are secure because it is stored in binary format/encrypted form and it gets decrypted at server.
4. Session is independent for every client i.e. individual for every client.
5. There is no limitation on size or number of sessions to be used in an application.
6. For all conditions/situations we can use sessions.
7. We cannot disable the sessions. Sessions can be used without cookies too (by disabling cookies).
8. The disadvantage of session is that it is a burden/overhead on server.
9. Sessions are called as Non-Persistent cookies because its lifetime can be set manually.

PHP Date

While PHP's date() function have an overwhelming amount of options available.

PHP date - the timestamp

The date function always formats a timestamp, whether you supply one or not.

- Timestamp: A timestamp is the number of seconds from January 1, 1970 at 00:00.

What time is it?

The date function uses letters of the alphabet to represent various parts of a typical date and time format.

- d: The day of the month. The type of output you can expect is 01 through 31.
- m: The current month, as a number. You can expect 01 through 12.
- y: The current year in two digits ##. You can expect 00 through 99

However, other characters like a slash "/" can be inserted between the letters to add additional formatting.

PHP Code:

```
<?php echo date("m/d/Y");  ?>
```

Display:

```
12/20/2012
```

Supplying a timestamp

This example below uses the mktime function to create a timestamp for tomorrow. To go one day in the future we simply add one to the day argument of mktime.

Note: These arguments are all optional. If you do not supply any arguments the current time will be used to create the timestamp.

- mktime(hour, minute, second, month, day, year, daylight savings time)

PHP Code:

```
<?php
	$tomorrow = mktime(0, 0, 0, date("m"), date("d")+1, date("Y"));
	echo "Tomorrow is ".date("m/d/Y", $tomorrow);
?>
```

Notice that we used one letter at a time with the function date to get the month, day and year. For example the date("m") will return the month's number 01-12.

Display:

```
Tomorrow is 12/21/2012
```

PHP date

Important Full Date and Time:

- r: Displays the full date, time and timezone offset. It is equivalent to manually entering date("D, d M Y H:i:s O")

Time:

- a: am or pm depending on the time
- A: AM or PM depending on the time
- g: Hour without leading zeroes. Values are 1 through 12.
- G: Hour in 24-hour format without leading zeroes. Values are 0 through 23.

- h: Hour with leading zeroes. Values 01 through 12.
- H: Hour in 24-hour format with leading zeroes. Values 00 through 23.
- i: Minute with leading zeroes. Values 00 through 59.
- s: Seconds with leading zeroes. Values 00 through 59.

Day:

- d: Day of the month with leading zeroes. Values are 01 through 31.
- j: Day of the month without leading zeroes. Values 1 through 31
- D: Day of the week abbreviations. Sun through Sat
- l: Day of the week. Values Sunday through Saturday
- w: Day of the week without leading zeroes. Values 0 through 6.
- z: Day of the year without leading zeroes. Values 0 through 365.

Month:

- m: Month number with leading zeroes. Values 01 through 12
- n: Month number without leading zeroes. Values 1 through 12
- M: Abbreviation for the month. Values Jan through Dec
- F: Normal month representation. Values January through December.
- t: The number of days in the month. Values 28 through 31.

Year:

- L: 1 if it's a leap year and 0 if it isn't.
- Y: A four digit year format
- y: A two digit year format. Values 00 through 99.

Other Formatting:

- U: The number of seconds since the Unix Epoch (January 1, 1970)
- O: This represents the Timezone offset, which is the difference from Greenwich Mean Time (GMT). 100 = 1 hour, -600 = -6 hours

Complete Date Reference: http://php.net/manual/en/function.date.php

Chapter 3

XML

What is XML?

- XML: e**X**tensible **M**arkup **L**anguage.
- XML is a standard for creating markup languages which describe the structure of data.
- It is a widely used system for defining data formats.
- XML is cross-platform, independent tool for exchanging data.
- XML is a **framework** for defining markup languages:

1. There is **no fixed collection of markup tags** - we may define our own tags, tailored for our kind of information
2. Each XML language is targeted at its own application domain, but the languages will share many features
3. There is a common set of **generic tools** for processing documents
4. is a markup language which relies on the concept of rule-specifying tags and the use of a tag-processing application that knows how to deal with the tags.

XML Basics:

- Developed by W3C (World Wide Web Consortium).
- It is the pared-down version of SGML (Standardized Generalized Markup Language).
- Designed especially for web documents
- Allows you to create your own language for displaying documents.
- "***XML is not the replacement of HTML.***"
- XML requires a processing application. There is no XML browser in market yet.
- XML isn't about display – it's about Structure.
- XML file should be saved with the extension "**.xml**".
- You can use any text editor to write XML code. And the output can be viewed in any browser.
- Every XML page consists of processing instruction at the very first line.
- There should be a root element in every XML document, which should be unique.

Official Goals of XML

“Self-education, the only kind of education that exists.”

Rules for writing XML:

- The first line must be **<?XML version = “1.0”?>**.
- Tags are enclosed in angle brackets.
- Tags are case sensitive and must have a matching closing tag.
- Tags may contain attributes in the form **name = “value”**.
- Tags may contain text, other tags, or both. Tags content lies between the starting and ending tag.

Advantages of XML:

- The first benefit of XML is that because you are writing your own markup language, you are not restricted to a limited set of tags defined by proprietary vendors.
- With XML, you can create your own set of tags at your own pace.
- XML allows every person/organization to build their own tag library which suits their needs perfectly.
- XML allows you to define all sorts of tags with all sorts of rules, such as tags representing business rules or tags representing data description or data relationships.
- With XML, GUI is extracted. Thus, changes to display do not require futzing with the data. Instead separate style-sheet will specify a table display or a list display.
- Searching the data is easy and efficient. Search-engine can simply parse the description-bearing tags rather than muddling in the data. Tags provide the search engines with the intelligence they lack.
- Complex relationships like tree and inheritance can be communicated.
- The code is much more legible to the person coming into the environment with no prior knowledge.

Disadvantages of XML:

- XML requires a processing application.
- There are no XML browsers on the market yet. Thus, XML documents must either be converted into HTML before distribution or converting it to HTML on-the-fly by middleware.
- Barring translation, developers must code their own processing applications.
- XML isn't about display -- it's about structure.

- This has implications that make the browser question secondary. So the whole issue of what is to be displayed and by what means is intentionally left to other applications.

XML Declaration:

- To begin an XML document, it is a good idea to include the XML declaration as the very first line of the document.
- The XML declaration is a processing instruction that notifies the processing agent that the following document has been marked up as an XML document.
- The XML declaration looks as following:

  <?xml version = "1.0"?>
- In its full regalia, the XML declaration might look like the following:

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

Elements:

- Elements are the basic unit of XML content.
- Syntactically, an element consists of a start tag, and an end tag, and everything in between.
- *Example:*

  <NAME>Frank Lee</NAME>
- All XML documents must have at least one root element to be well formed. The root element, also often called the document tag, must follow the prolog (XML declaration plus DTD) and must be a nonempty tag that encompasses the entire document.
- XML defines the text between the start and end tags to be "character data" and the text within the tags to be "markup".

WELL-FORMED XML

A well-formed XML Document has to follow several generic rules associated with element and attributes.

Rules for Elements:

- Every starting tag must have a matching end tag.

- Tags can't overlap.
- XML documents can have only one root element.
- Element names must XML naming conventions.
- XML is case sensitive.
- XML will keep white space in your text.

Attributes:

- Attributes are properties that are associated with an element.
- An attribute specification is the "*name-value*" pair that is associated with the element.
- Adding attribute provides an alternative way to include information in an element.
- Like XML tags, attributes are also not restricted to store the type of information.
- They are attached with a starting tag.
- Attributes must have values.
- We can use double quote (" ") or single quote (' ') to delineate attribute values.
- *Example:*

  <ELEMENT_NAME attribute_name = "value"> or

  < ELEMENT_NAME attribute_name = 'value'>

Why use Attribute?

- Attributes provide meta-data that may not be relevant to most applications dealing with our XML.
- Attributes are easier to use, they don't require nesting and you don't have to worry about crossed tags.
- Attributes takes much less space than elements.

Rules for Creating Attributes:

- The name must begin with a letter or an underscore ('_'), followed by '0' or some letters, digits, periods ('.'), hyphens('-').

- The XML specification states that attribute names beginning with the prefix “xml” (in any combination of uppercase or lowercase letters) are “reserved for standardization”. Although IE-5 doesn’t enforce this restriction, its better not to use this prefix to avoid future problems.
- A particular attribute name can appear only once in the same start-tag or empty tag. Example: **<ANIMATION FileName**=”a.ani” **FileName**=”b.ani”**>** Some Text **</ANIMATION>**
- The value of attribute must be delimited using either single quotes (‘ ‘) or double quotes (“ “).
- The value of attribute cannot contain the same quote character used to delimit it.
- The value of attribute cannot include the ‘<’ character and ‘&’ character except to begin a character or entity reference.

Comments:

- Comments provide a way to insert text that isn’t really part of the document, but rather is intended for people who are reading the XML source itself.
- Comments Start with the String <!- - and end with the string - ->
- *Example:*

<NAME NickName = “Maddy”**>**

    <FIRST-NAME> John **<FIRST-NAME>**

    <MIDDLE-NAME> </MIDDLE-NAME>

        <!- - John lost his Middle Name in Fire - - >

    <LAST-NAME> Doe **</LAST-NAME>**

</NAME>

- ***Points to Ponder:***

- You cannot have comment inside a tag.

<MIDDLE-NAME> </MIDDLE-NAME<!- - John lost his Middle Name in Fire - - > **>**

- You can’t use ‘- -‘ character inside a comment.

Escaping Characters:

- There are some reserved characters that you can’t include in your PCDATA because they are used in XML syntax.

- For Example:
  '**<**' and '**&**' characters are example of those characters.

  <COMPARISON> 6 is < 7 & 7 > 6 **</COMPARISION>**
- To escape such characters, you simply can use &lt; and &amp; characters.
- It automatically un-escapes the characters for you when it displays the document.

Character Reference:

- The strings such as &#NNN; (where 'NNN' is the Unicode numbers), used to escape the reserved words of XML is known as C*haracter Reference*.
- It can also be &#Xnnn; with 'X' preceding the number and 'nnn' is a hexadecimal number.

Entity Reference:

- The strings such as &lt; used to escape the reserved words of XML is known as *Entity Reference*.
- The Entity Reference can be like &gt; &amp; &lt; etc.
- You can either use character reference or entity reference to escape the characters.

CDATA:

- If you have a lot of '<' and '>' characters that you need to be displayed in XML document, then it is wise to use CDATA sections.
- CDATA is another inherited term from SGML. It stands for Character Data.
- In the special case of CDATA blocks, all tags and entity references are ignored by an XML processor that treats them just like any old character data.
- Using CDATA sections; we can tell the XML parser not to parse the text, until it gets the end of the section.
- CDATA blocks have been provided as a convenience measure when you want to include large blocks of special characters a character data, but you do not want to have to use entity references all the time.
- To avoid the inconvenience of translating all special characters, you can use a CDATA block to specify that all character data should be considered character data whether or not it "looks" like a tag or entity reference.

- *Where to place the CDATA section:*

- CDATA should not be placed above the root element.
- CDATA should not be placed within the content of document element.

- ***Example of CDATA:***

```
<EXAMPLE>
<![CDATA[
  <COMPARISION>
          6 is < 7 & 7 > 6.
  </ COMPARISION >
]]>
</EXAMPLE>
```

Processing Instructions:

- A processing instruction is a bit of information meant for the application using the XML document.
- That is, they are not really of interest to the XML parser.
- Instead, the instructions are passed intact straight to the application using the parser.
- The application can then pass this on to another application or interpret it itself.
- All processing instructions follow the generic format of:

  <?NAME_OF_APPLICATION Instructions?>
- Example of processing instructions:

```
<?XML version = "1.0"?>
  <doc>
          <?JavaScript function funct1 (y){ ?>
          <?JavaScript var x;?>
          <?JavaScript x = y * 10;?>
          <?JavaScript return (x);?>
          <?JavaScript }?>
  </doc>
```

Document Type Definitions

What is DTD?

- DTD defines the rules that set out how the document should be structured, what elements should be included, what kind of data may be included and what default values to use.
- Multiple documents and applications can share DTDs.
- DTDs use a formal grammar to describe the structure and syntax of an XML document, including the permissible values for much of that document's content.
- DTDs:
  1. Provide a formal and complete definition of an XML vocabulary.
  2. Are shareable descriptions of the structure of an XML documents.
  3. Are a way to validate specific instances of XML documents and constraints.
  4. Are restricted to one DTD per document instance.
  5. Specifies the validity of each tag.

Internal Vs. External DTD

- DTD may be divided into two parts: the internal subset and the external subset.
- These subsets are relative to the document instance.
- ▪ The internal subset is a portion of the DTD including within the document.
- ▪ The external subset is a portion of declarations that are located in a separate document.
- A DTD might be contained entirely within the document, with no external subset, or a document may simply refer to an external subset and contain no DTD declaration of its own.
- In many cases, DTD may use a combination of both.
- DTD declarations in the internal subset have priority over those in the external subset.

Associating a DTD with an XML document

- Each XML document can be associated with one, and only one DTD using single DOCTYPE declaration.
- DTDs are linked to XML documents using markup called the Document Type Definitions.

- This declaration is commonly referred to as "the DOCTYPE declaration" to differentiate it from a DTD.

The Document Type (DOCTYPE) Declarations:

- A document type declaration is placed in an XML document's prolog to say what DTD that document adheres to.
- It also specifies which element is the root element of the document.
- A document type *declaration* is not the same thing as a document type *definition*.
- A document type declaration must contain or refer to a document type definition, but a document type definition never contains a document type declaration.
- A document type declaration begins with <!DOCTYPE and ends with a >.
- A document type declaration has this basic form:

<!DOCTYPE *name_of_root_element*
  SYSTEM "*URL of the external DTD subset*" [
  internal DTD subset
]>

- Here *name_of_root_element* is simply the name of the root element.
- The SYSTEM keyword indicates that what follows is a URL where the DTD is located.
- The square brackets enclose the internal subset of the DTD—that is, those declarations included inside the document itself.
- The DOCTYPE declaration consists of:

1. The usual XML tag delimiters ("<" and "?").
2. The exclamation mark ("!") that signifies a special XML declaration.
3. The DOCTYPE keyword.
4. The name of the document element (document_element).
5. One of two legal source keywords.
6. One of two DTD locations to associate an external DTD subset within a document.
7. Some additional declarations referring to the internal subset of the DTD.

Validating Against a DTD

- To be considered *valid,* an XML document must satisfy four criteria:

  1. It must be well formed.

  2. It must have a document type declaration.

  3. Its root element must be the one specified by the document type declaration.

  4. It must satisfy all the constraints indicated by the DTD specified by the document type declaration.

The Document Element Name

- The first variable of any DOCTYPE declaration is the name of the document element.
- This is required to be the root element of XML document.
- **Example:**

<?XML version = "1.0"?>

<!DOCTYPE Employee………..>

<Employee>

</Employee>

Basic DTD Declarations

- DTD declarations are delimited with the usual XML tag delimiters ("<" and ">").
- Like DOCTYPE declarations, all DTD declarations are indicated by the use of the exclamation mark ("!") followed by a keyword, and its specific parameters

  <! Keyword parameter1, paramenter2, …………, parameterN>
- There are four basic keywords used in DTD declarations

1. ELEMENT
2. ATTLIST
3. ENTITY
4. NOTATION

Element Type (ELEMENT) Declarations:

- Elements are described using the element type declaration.
- This declaration can have one of two different forms depending on the value of the category parameter

<!ELEMENT name Category>
<!ELEMENT name (Content_Model)>

Element content Categories

- There are 5 categories of element content:

Content Category	**Description**
ANY	Element type may contain any well formed XML data.
EMPTY	Element type may contain any text or child elements- only elements attributes are permitted.
Element	Element type contains only child elements no additional text is permitted.
Mixed	Element type may contain text and/or child element.
PCDATA	Element type may contain text (character data) only.

Content Models

- Content models are used to describe the structure and content of a given element type.
- The content may be:
  1. Character data (PCDATA content).
  2. One or more child element types (element-only content).
  3. A combination of the two (mixed content).

- The key difference between element content and mixed content is the use of the **#PCDATA** keyword.
- If present, the content model is either mixed or PCDATA.
- The absence of this keyword indicates element-only content.

Cardinality

- Cardinality operators define how many child elements may appear in a content model.
- There are four cardinality operators:

Operators	Description
[none]	The absence of a cardinality operator character indicates that one, and only one, instance of child element is allowed (required).
?	Zero or one element – optional singular element.
*	Zero or more element – optional element(s).
+	One or more child elements – required element(s).

/************* Example of Cardinality Operators ***************/

<!ELEMENT PersonName**(**
  (Mr./Miss) **?**, First_Name **+**, Middle_Name *****, Last_Name**)>**

The Attribute (ATTLIST) Declarations

- Attributes can be used to describe the meta-data or properties of the associated element.
- Element attributes are described using the attribute list declarations, also called ATTLIST declarations.
- This declaration has the usual DTD declarations format, using the ATTLIST keyword plus zero or more attribute definitions.

<! ATTLIST element_name attrName attrType attrDefault defaultValue>

The value is a predefined xml value. The default-value can be one of the following:

'Value'	-	The default value of the attribute
#REQUIRED	-	The attribute is required
#IMPLIED	-	The attribute is not required

#FIXED value - The attribute value is fixed

/************* Example of Attribute List Declarations ************/

```
<!ELEMENT Book EMPTY>
<!ATTLIST Book
   isbn     CDATA  #REQUIRED
   title    CDATA  #REQUIRED
   author  CDATA  #REQUIRED
   price   CDATA  #IMPLIED (optional)
>
```

Attribute Types

- There are 10 different types of attributes defined in XML 1.0 recommendation.

Attribute	**Description**
CDATA	Character Data (simple text string)
Enumerated values (Choice list)	Attribute must be one of a series that is explicitly defined in DTD.
ID	Attribute value is the unique identifier for this element instance
IDREF	A reference to the element with an ID attribute that has the same value as that of IDREF
IDREFS	A list of IDREFs delimited by white space
NMTOKEN	A name token – a text string that confirms to the XML name rules
NMTOKENS	A list of NMTOKENs delimited by white spaces
ENTITY	The name of a pre-defined entity
ENTITIES	A list of ENTITY name delimited by white spaces
NOTATION	Attribute value must be a notation type that is explicitly declared elsewhere in the DTD

XML Elements vs. Attributes

In XML, there are no rules about when to use attributes, and when to use child elements.

Use of Elements vs. Attributes

Data can be stored in child elements or in attributes.

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

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

In the first example gender is an attribute. In the last, gender is a child element. Both examples provide the same information. There are no rules about when to use attributes, and when to use child elements. My experience is that attributes are handy in HTML, but in XML you should try to avoid them. Use child elements if the information feels like data.

I like to store data in child elements.

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

```
<note date="12/11/2002">
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

A date element is used in the second example:

```
<note>
        <date>12/11/2002</date>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

An expanded date element is used in the third:

```
<note>
<date>
  <day>12</day>
  <month>11</month>
  <year>2002</year>
</date>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Should you avoid using attributes?

Some of the problems with attributes are:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a DTD

DTD – Entities

Entities are variables used to define shortcuts to standard text or special characters. Entity references are references to entities. Entities can be declared internal or external.

An Internal Entity Declaration

Syntax

```
<!ENTITY entity-name "entity-value">
```

Example DTD:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

XML example:

```
<author>&writer;&copyright;</author>
```

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration

Syntax<!ENTITY entity-name SYSTEM "URI/URL">

DTD Example:

```
<!ENTITY writer SYSTEM "http://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.w3schools.com/entities.dtd">
```

XML example:

```
<author>&writer;&copyright;</author>
```

/**************************** Example of DTD ***************************/**

```
<!DOCTYPE Employees[
<!ELEMENT Employees (employee +)>
<!ELEMENT employee (Name +, Position +, Address +)>
<!ATTLIST employee id ID #REQUIRED>
<!ELEMENT Name (First +, Middle ?, Last +)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Middle(#PCDATA)>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT Position (#PCDATA)>
<!ELEMENT Address (Street ?, City +, State +, Zip *)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Zip (#PCDATA)>
]>
```

Limitations of DTD

- Some limitations of DTD include:

1. DTD are not extensible, unlike XML itself.
2. Only one DTD may be associated with each XML document.
3. DTDs do not work well with XML namespaces.
4. Supports very weak data typing.
5. Limited content model descriptions.
6. No object oriented type object inheritance.
7. A document can override / ignore an external DTD using internal subset.
8. Non-XML syntax.
9. No DOM support.
10. Relatively few older, more expensive tools.
11. Very limited support to modularity and reuse.
12. Too simple ID attribute mechanism (no points to requirements, uniqueness scope etc.)

DTD - Examples from the Cloud

TV Schedule DTD

```
<!DOCTYPE TVSCHEDULE [
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

Newspaper Article DTD

```
 <!DOCTYPE NEWSPAPER [
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>
<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">
]>
```

Product Catalog DTD

```
<!DOCTYPE CATALOG [
<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">
<!ELEMENT CATALOG (PRODUCT+)>
<!ELEMENT PRODUCT
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>
<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">
<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>
        <!ELEMENT NOTES (#PCDATA)>
        ]>
```

-wts

XML Schema (XSD)

What is Schema?

- XML Schema is an XML-based alternative to DTDs.
- An XML Schema describes the structure of an XML document.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema defines:

- elements that can appear in a document
- attributes that can appear in a document
- which elements are child elements
- the order of child elements
- the number of child elements
- whether an element is empty or can include text
- data types for elements and attributes
- default and fixed values for elements and attributes

'XML Schemas are the Successors of DTDs'

We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types

- XML Schemas support namespaces
- XML Schema is a W3C Standard

'XML Schema became a W3C Recommendation 02. May 2001.' 'XML Schemas are much more powerful than DTDs.'

XML Schemas Support Data Types

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

With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- 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. *Some benefits of that XML Schema 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 Schema with the XML DOM
- You can transform your Schema with XSLT
- XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3, November and in other countries as 11, March. However, XML elements with a data type like this: <date type="date">2004-03-11</date> Ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

'XML Schemas are extensible, because they are written in XML.'

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Well-Formed XML

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted

- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

XML Namespace

XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace, the ambiguity between identically named elements or attributes can be resolved.

A simple example would be to consider an XML instance that contained references to a customer and an ordered product. Both the customer element and the product element could have a child element named id. References to the id element would therefore be ambiguous; placing them in different namespaces would remove the ambiguity.

For example, the following declaration maps the "xhtml:" prefix to the XHTML namespace:

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

There are really two fundamental needs for namespaces:

1. To disambiguate between two elements that happen to share the same name
2. To group elements relating to a common idea together

[*SELF STUDY FOR MORE*]

A Simple XML Document

Look at this simple XML document called "note.xml":

<?xml version="1.0"?>

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

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

</note>

A DTD File

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The first line defines the note element to have four child elements: "to, from, heading, body".
Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
 <?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<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>
</xs:schema>
```

The note element is a complex type because it contains other elements. The other elements (to, from, heading, body) are simple types because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to a DTD

This XML document has a reference to a DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.w3schools.com/dtd/note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

'The <schema> element is the root element of every XML Schema.'

The <schema> Element

The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?>
<xs:schema>
```

```
...
</xs:schema>
```

The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
...
</xs:schema>
```

The following fragment:xmlns:xs="http://www.w3.org/2001/XMLSchema"

Indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

This fragment:targetNamespace="http://www.w3schools.com"

Indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

This fragment:xmlns="http://www.w3schools.com"

Indicates that the default namespace is "http://www.w3schools.com".

This fragment:elementFormDefault="qualified"

Indicates that any elements used by the XML instance document, which were declared in this schema must be namespace qualified.

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The following fragment:xmlns="http://www.w3schools.com"
Specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3schools.com" namespace.

Once you have the XML Schema Instance namespace available:
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance you can use the schemaLocation attribute.

And next is the location of the XML schema to use for that namespace:
xsi:schemaLocation=http://www.w3schools.com note.xsd

What is a Simple Element?
A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

You can add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element
The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

Where xxx is the name of the element and yyy is the data type of the element.

The most common data types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example:

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

And

Here are the corresponding simple element definitions:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified. A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

```
<xs:element name="color" type="xs:string" default="red"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

How to Define an Attribute?

'All attributes are declared as simple types.'

The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

Where xxx is the name of the attribute and yyy specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

Default and Fixed Values for Attributes

Attributes may have a default value OR a fixed value specified.

In the following example the default value is "EN":

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

In the following example the fixed value is "EN":

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Restrictions on Content

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets. Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint. The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The example above could also have been written like this:

```
<xs:element name="car" type="carType"/>
<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

'In this case the type "carType" can be used by other elements because it is not a part of the "car" element.'

Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][A-Z][A-Z]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example also defines an element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:

```
<xs:element name="initials">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[xyz]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="prodid">
<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Other Restrictions on a Series of Values

The example below defines an element called "letter" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:

```
<xs:element name="letter">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-z])*"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example also defines an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "stop":

```
<xs:element name="letter">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-z][A-Z])+"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

```
<xs:element name="gender">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="male|female"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9]{8}"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.

This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

```
<xs:element name="address">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="replace"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

```
<xs:element name="address">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Restrictions on Length

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints. *This example defines an element called "password" with a restriction. The value must be exactly eight characters:*

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:length value="8"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:minLength value="5"/>
    <xs:maxLength value="8"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Restrictions for Data types

- ***Enumeration -*** *Defines a list of acceptable values.*
- ***fractionDigits -*** *Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero.*
- ***length*** *- Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero.*
- ***maxExclusive -*** *Specifies the upper bounds for numeric values (the value must be less than this value).*
- ***maxInclusive -*** *Specifies the upper bounds for numeric values (the value must be less than or equal to this value).*
- ***maxLength -*** *Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero.*
- ***minExclusive -*** *Specifies the lower bounds for numeric values (the value must be greater than this value).*
- ***minInclusive -*** *Specifies the lower bounds for numeric values (the value must be greater than or equal to this value).*
- ***minLength*** *- Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero.*
- ***pattern*** *- Defines the exact sequence of characters that are acceptable.*
- ***totalDigits -*** *Specifies the exact number of digits allowed. Must be greater than zero.*
- ***whitespace -*** *Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled.*

XSD Complex Elements

A complex element contains other elements and/or attributes.

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.
There are four kinds of complex elements:

1. empty elements
2. elements that contain only other elements
3. elements that contain only text
4. elements that contain both other elements and text

'Each of these elements may contain attributes as well!'

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description> It happened on <date lang="norwegian">03.03.99</date> ....</description>
```

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

If you use the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

XSD Complex Empty Elements

An empty complex element cannot have contents, only attributes.

Complex Empty Elements

An empty XML element: <product prodid="1345" />

The "product" element above has no content at all. To define a type with no content, we must define a type that allows only elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

In the example above, we define a complex type with a complex content. The **complexContent** element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

However, it is possible to declare the "product" element more compactly, like this:

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Or

you can give the complexType element a name, and let the "product" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="product" type="prodtype"/>
<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

XSD Complex Type - Elements Only

An "elements-only" complex type contains an element that contains only other elements.

An XML element, "person", that contains only other elements:

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

You can define the "person" element in a schema, like this:

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

XSD Complex Text-Only Elements

A complex text-only element can contain text and attributes.

Complex Text-Only Elements

This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element, like this:

```
<xs:element name="somename">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="basetype"> .... </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

OR

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype"> ... </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

'Use the extension/restriction element to expand or to limit the base simple type for the element.'

Here is an example of an XML element, "shoesize", that contains text-only:

```
<shoesize country="france">35</shoesize>
```

The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "country":

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XSD Complex Types with Mixed Content

A mixed complex type element can contain attributes, elements, and text.

Complex Types with Mixed Content

An XML element, "letter", that contains both text and other elements:

```
<letter>
Dear Mr.<name>John Smith</name>.
Your order <orderid>1032</orderid>
will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

The following schema declares the "letter" element:

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Note: To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true".

XSD Complex Types Indicators

We can control HOW elements are to be used in documents with indicators.

Order indicators:

1. All
2. Choice
3. Sequence

Occurrence indicators:

1. maxOccurs
2. minOccurs

Group indicators:

1. Group name
2. attributeGroup name

All Indicators

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Sequence Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement.

***A working example:* An XML file called "Myfamily.xml":**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
<person>
        <full_name>Hege Refsnes</full_name>
        <child_name>Cecilie</child_name>
</person>
<person>
        <full_name>Tove Refsnes</full_name>
        <child_name>Hege</child_name>
        <child_name>Stale</child_name>
        <child_name>Jim</child_name>
        <child_name>Borge</child_name>
</person>
<person>
    <full_name>Stale Refsnes</full_name>
</person>
</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="persons">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="person" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      minOccurs="0" maxOccurs="5"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Group Indicators

Group indicators are used to define related sets of elements.

Element Groups

Element groups are defined with the group declaration, like this:

```
<xs:group name="groupname">
 ...
</xs:group>
```

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

After you have defined a group, you can reference it in another definition, like this:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

```
<xs:attributeGroup name="groupname">
  ...
</xs:attributeGroup>
```

The following example defines an attribute group named "personattrgroup":

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

XSD The <any> Element

The <any> element enables us to extend the XML document with elements not specified by the schema!

The <any> Element

The <any> element enables us to extend the XML document with elements not specified by the schema.

The following example is a fragment from an XML schema called "family.xsd". It shows a declaration for the "person" element. By using the <any> element we can extend (after <lastname>) the content of "person" with any element:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now we want to extend the "person" element with a "children" element. In this case we can do so, even if the author of the schema above never declared any "children" element.
Look at this schema file, called "children.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="children">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="childname" type="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The XML file below (called "Myfamily.xml"), uses components from two different schemas; "family.xsd" and "children.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com children.xsd">
<person>
<firstname>Hege</firstname>
<lastname>Refsnes</lastname>
```

```
<children>
            <childname>Cecilie</childname>
</children>
</person>
<person>
<firstname>Stale</firstname>
<lastname>Refsnes</lastname>
</person>
</persons>
```

The XML file above is valid because the schema "family.xsd" allows us to extend the "person" element with an optional element after the "lastname" element.

The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

XSD The <anyAttribute> Element

The <anyAttribute> element enables us to extend the XML document with attributes not specified by the schema!

The following example is a fragment from an XML schema called "family.xsd". It shows a declaration for the "person" element. By using the <anyAttribute> element we can add any number of attributes to the "person" element:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

Now we want to extend the "person" element with a "gender" attribute. In this case we can do so, even if the author of the schema above never declared any "gender" attribute.

Look at this schema file, called "attribute.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:schema>
```

The XML file below (called "Myfamily.xml"), uses components from two different schemas; "family.xsd" and "attribute.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com attribute.xsd">
<person gender="female">
        <firstname>Hege</firstname>
        <lastname>Refsnes</lastname>
</person>
<person gender="male">
        <firstname>Stale</firstname>
        <lastname>Refsnes</lastname>
</person>
</persons>
```

The XML file above is valid because the schema "family.xsd" allows us to add an attribute to the "person" element. The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

XSD Element Substitution

With XML Schemas, one element can substitute another element.

Element Substitution

Suppose, we have users from two different countries: England and Norway. We would like the ability to let the user choose whether he or she would like to use the Norwegian element names or the English element names in the XML document.

To solve this problem, we could define a substitutionGroup in the XML schema. First, we declare a head element and then we declare the other elements, which state that they are substitutable for the head element.

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

In the example above, the "name" element is the head element and the "navn" element is substitutable for "name".

Look at this fragment of an XML schema:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

A valid XML document (according to the schema above) could look like this:

```
<customer>
  <name>John Smith</name>
</customer>
```

Or

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

What are Global Elements?

Global elements are elements that are immediate children of the "schema" element! Local elements are elements nested within other elements.

An XSD Example

This chapter will demonstrate how to write an XML Schema. You will also learn that a schema can be written in different ways.

An XML Document

Let's have a look at this XML document called "shiporder.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
 <orderperson>John Smith</orderperson>
 <shipto>
  <name>Ola Nordmann</name>
  <address>Langgt 23</address>
  <city>4000 Stavanger</city>
  <country>Norway</country>
 </shipto>
 <item>
  <title>Empire Burlesque</title>
  <note>Special Edition</note>
  <quantity>1</quantity>
  <price>10.90</price>
 </item>
 <item>
  <title>Hide your heart</title>
  <quantity>1</quantity>
  <price>9.90</price>
 </item>
</shiporder>
```

The XML document above consists of a root element, "shiporder", that contains a required attribute called "orderid". The "shiporder" element contains three different child elements: "orderperson", "shipto" and "item". The "item" element appears twice, and it contains a "title", an optional "note" element, a "quantity", and a "price" element.

Create an XML Schema

Here is the complete listing of the schema file called "shiporder.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="orderperson" type="xs:string"/>
   <xs:element name="shipto">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="orderid" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
</xs:schema>
```

XSD Date and Time Data Types

Date and time data types are used for values that contain date and time.

Date Data Type

The date data type is used to specify a date. The date is specified in the following form "YYYY-MM-DD" where:

YYYY indicates the year

MM indicates the month

DD indicates the day

Note: All components are required!

The following is an example of a date declaration in a schema:

```
<xs:element name="start" type="xs:date"/>
```

An element in your document might look like this:

```
<start>2002-09-24</start>
```

Time Zones

To specify a time zone, you can either enter a date in UTC time by adding a "Z" behind the date - like this:

```
<start>2012-09-24Z</start>
```

Or

You can specify an offset from the UTC time by adding a positive or negative time behind the date - like this:

```
<start>2012-09-24-06:00</start>
```

or

```
<start>2012-09-24+06:00</start>
```

Time Data Type

The time data type is used to specify a time. The time is specified in the following form "hh:mm:ss" where:

hh indicates the hour

mm indicates the minute

ss indicates the second

Note: All components are required!

The following is an example of a time declaration in a schema:

```
<xs:element name="start" type="xs:time"/>
```

An element in your document might look like this:

```
<start>09:00:00</start>
```

Or

It might look like this:

```
<start>09:30:10.5</start>
```

Time Zones

To specify a time zone, you can either enter a time in UTC time by adding a "Z" behind the time - like this:

```
<start>09:30:10Z</start>
```

Or

You can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<start>09:30:10-06:00</start>
```

Or

```
<start>09:30:10+06:00</start>
```

DateTime Data Type

The dateTime data type is used to specify a date and a time. The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss" where:

YYYY indicates the year

MM indicates the month

DD indicates the day

T indicates the start of the required time section

hh indicates the hour

mm indicates the minute

ss indicates the second

Note: All components are required!

The following is an example of a dateTime declaration in a schema:

```
<xs:element name="startdate" type="xs:dateTime"/>
```

An element in your document might look like this:

```
<startdate>2002-05-30T09:00:00</startdate>
```

Or

It might look like this:

```
<startdate>2002-05-30T09:30:10.5</startdate>
```

Time Zones

To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" behind the time - like this:

```
<startdate>2002-05-30T09:30:10Z</startdate>
```

Or

You can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<startdate>2002-05-30T09:30:10-06:00</startdate>
```

Or

```
<startdate>2002-05-30T09:30:10+06:00</startdate>
```

Duration Data Type

The duration data type is used to specify a time interval.

The time interval is specified in the following form "PnYnMnDTnHnMnS" where:

P indicates the period (required)

nY indicates the number of years

nM indicates the number of months

nD indicates the number of days

T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds)

nH indicates the number of hours

nM indicates the number of minutes

nS indicates the number of seconds

The following is an example of a duration declaration in a schema:

```
<xs:element name="period" type="xs:duration"/>
```

An element in your document might look like this:

```
<period>P5Y</period>
```

The example above indicates a period of five years.

Or

It might look like this:

```
<period>P5Y2M10D</period>
```

The example above indicates a period of five years, two months, and 10 days.

Or

It might look like this:

```
<period>P5Y2M10DT15H</period>
```

The example above indicates a period of five years, two months, 10 days, and 15 hours.

Or

It might look like this :

```
<period>PT15H</period>
```

The example above indicates a period of 15 hours.

Negative Duration

To specify a negative duration, enter a minus sign before the P:

```
<period>-P10D</period>
```

The example above indicates a period of minus 10 days.

Date and Time Data Types

***Date* -** Defines a date value

***dateTime* -** Defines a date and time value

duration - Defines a time interval

gDay - Defines a part of a date - the day (DD)

***gMonth* -** Defines a part of a date - the month (MM)

***gMonthDay* -** Defines a part of a date - the month and day (MM-DD)

gYear - Defines a part of a date - the year (YYYY)

***gYearMonth* -** Defines a part of a date - the year and month (YYYY-MM)

time - Defines a time value

Restrictions on Date Data Types

Restrictions that can be used with Date data types:

- ***enumeration***
- ***maxExclusive***
- ***maxInclusive***
- ***minExclusive***
- ***minInclusive***
- ***pattern***
- ***whitespace***

XSD Numeric Data Types

Decimal data types are used for numeric values.

Decimal Data Type

The decimal data type is used to specify a numeric value. The following is an example of a decimal declaration in a schema:

```
<xs:element name="prize" type="xs:decimal"/>
```

An element in your document might look like this:

```
<prize>999.50</prize>
```

Or

prize>+999.5450</prize>

Or

<prize>-999.5230</prize>

Or

<prize>0</prize>

Or

<prize>14</prize>

Note: The maximum number of decimal digits you can specify is 18.

Integer Data Type

The integer data type is used to specify a numeric value without a fractional component. The following is an example of an integer declaration in a schema:

<xs:element name="prize" type="xs:integer"/>

An element in your document might look like this:

<prize>999</prize>

Or

<prize>+999</prize>

Or

<prize>-999</prize>

Or

<prize>0</prize>

Numeric Data Types

Note that all of the data types below derive from the Decimal data type (except for decimal itself)!

Byte **-** A signed 8-bit integer

Decimal **-** A decimal value

Int - A signed 32-bit integer

Integer - An integer value

Long - A signed 64-bit integer

negativeInteger - An integer containing only negative values (.., -2, -1.)

nonNegativeInteger - An integer containing only non-negative values (0, 1, 2, ..)

nonPositiveInteger - An integer containing only non-positive values (.., -2, -1, 0)

positiveInteger - An integer containing only positive values (1, 2, ..)

***short* -** A signed 16-bit integer

unsignedLong - An unsigned 64-bit integer

unsignedInt - An unsigned 32-bit integer

unsignedShort **-** An unsigned 16-bit integer

unsignedByte **-** An unsigned 8-bit integer

Restrictions on Numeric Data Types

Restrictions that can be used with Numeric data types:

- ***enumeration***
- ***fractionDigits***
- ***maxExclusive***
- ***maxInclusive***
- ***minExclusive***
- ***minInclusive***
- ***pattern***
- ***totalDigits***
- ***whitespace***

XSD Miscellaneous Data Types

Other miscellaneous data types are boolean, base64Binary, hexBinary, float, double, anyURI, QName, and NOTATION.

Boolean Data Type

The boolean data type is used to specify a true or false value. The following is an example of a boolean declaration in a schema:

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

An element in your document might look like this:

```
<prize disabled="true">999</prize>
```

Legal values for boolean are true, false, 1 (which indicates true), and 0 (which indicates false).

Binary Data Types

Binary data types are used to express binary-formatted data.

We have two binary data types:

- base64Binary (Base64-encoded binary data)
- hexBinary (hexadecimal-encoded binary data)

The following is an example of a hexBinary declaration in a schema:

<xs:element name="blobsrc" type="xs:hexBinary"/>

AnyURI Data Type

The anyURI data type is used to specify a URI. The following is an example of an anyURI declaration in a schema:

<xs:attribute name="src" type="xs:anyURI"/>.

An element in your document might look like this:

<pic src="http://www.w3schools.com/images/smiley.gif" />

Note: If a URI has spaces, replace them with %20.

Miscellaneous Data

- **anyURI**
- **base64Binary**
- **boolean**
- **double**
- **float**
- **hexBinary**
- **NOTATION**
- **QName**

Restrictions on Miscellaneous Data Types

Restrictions that can be used with the other data types:

- ***enumeration (a Boolean data type cannot use this constraint)***
- ***length (a Boolean data type cannot use this constraint)***
- ***maxLength (a Boolean data type cannot use this constraint)***
- ***minLength (a Boolean data type cannot use this constraint)***
- ***pattern***
- ***whitespace***

Displaying XML with CSS

With CSS (Cascading Style Sheets) you can add display information to an XML document. It is possible to use CSS to format an XML document.

Sample XML File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
 <CD>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
 </CD>
 <CD>
  <TITLE>Hide your heart</TITLE>
  <ARTIST>Bonnie Tyler</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS Records</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1988</YEAR>
 </CD>
.
.
.
</CATALOG>
```

cd_catalog.css

```
CATALOG { background-color: #ffffff; width: 100%; }
CD { display: block; margin-bottom: 30pt; margin-left: 0; }
TITLE { color: #FF0000; font-size: 20pt; }
ARTIST { color: #0000FF; font-size: 20pt; }
COUNTRY,PRICE,YEAR,COMPANY { display: block; color: #000000; margin-left: 20pt; }
```

'All CSS properties can be used for formatting. REFER 'CSS' FOR MORE.'

Formatting XML with CSS is not the most common method. W3C recommends using XSLT instead.

XSL for XML: a happy marriage!

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 Style sheet Language.

CSS = HTML Style Sheets
HTML uses predefined tags and the meanings of the tags are well understood. The <table> element in HTML defines a table - and a browser knows how to display it. Adding styles to HTML elements is simple. Telling a browser to display an element in a special font or color is easy with CSS.

XSL = XML Style Sheets
XML does not use predefined tags (we can use any tag-names we like), and the meaning of these tags are not well understood. A <table> element could mean an HTML table, a piece of furniture, or something else - and a browser does not know how to display it.

XSL describes how the XML document should be displayed! XSL is more Than a Style Sheet Language

1. XSLT - a language for transforming XML documents
2. XPath - a language for navigating in XML documents
3. XSL-FO - a language for formatting XML documents

XSLT - the language for transforming XML documents. XSLT is a language for transforming XML documents into XHTML documents or to other XML documents. XPath is a language for navigating in XML documents.

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 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 to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

How does it Works?

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 is a W3C Recommendation, it became a W3C Recommendation 16. November 1999.

XSLT Browsers

Nearly all-major browsers have support for XML and XSLT.

- Mozilla Firefox

- Google Chrome
- Safari
- Netscape
- Opera
- Internet Explorer

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="ISO-8859-1"?>
<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>
```

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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 align="left">Title</th>
      <th align="left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

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

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

XSLT <xsl:template> Element

An XSL style sheet consists of one or more set of rules that are called templates. Each template contains rules to apply when a specified node is matched.

The <xsl:template> Element

The <xsl:template> element is used to build templates. The match attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Since an XSL style sheet is an XML document itself, it always begins with the XML declaration:

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

The next element, <xsl:stylesheet>, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The <xsl:template> element defines a template. The match="/" attribute associates the template with the root of the XML source document. The content inside the <xsl:template> element defines some HTML to write to the output. The last two lines define the end of the template and the end of the style sheet.

The result of the transformation above will look like this:

```
My CD Collection
Title        Artist
.            .
```

The result from this example was a little disappointing, because no data was copied from the XML document to the output.

XSLT <xsl:value-of> Element

The <xsl:value-of> element is used to extract the value of a selected node. The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
     <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
     </tr>
   </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Note: The value of the select attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The result of the transformation above will look like this:

My CD Collection

Title	**Artist**
Empire Burlesque	Bob Dylan

XSLT <xsl:for-each> Element

The <xsl:for-each> element allows you to do looping in XSLT.

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

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

The value of the select attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The result of the transformation above will look like this:

My CD Collection

Title	**Artist**
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary More
Eros	Eros Ramazzotti
One night only	Bee Gees

Filtering the Output

We can also filter the output from the XML file by adding a criterion to the select attribute in the <xsl:for-each> element.

```
<xsl:for-each select="catalog/cd[artist='Bob Dylan']">
```

Legal filter operators are:

= (equal)

!= (not equal)

&lt; less than

&gt; greater than

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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[artist='Bob Dylan']">
      <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>
```

The result of the transformation above will look like this:

My CD Collection

Title	**Artist**
Empire Burlesque	Bob Dylan

XSLT <xsl:sort> Element

The <xsl:sort> element is used to sort the output.

To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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">
    <xsl:sort select="artist"/>
    <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>
```

The select attribute indicates what XML element to sort on.

The result of the transformation above will look like this:

My CD Collection

Title	**Artist**
Romanza	Andrea Bocelli
One night only	Bee Gees
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
The very best of	Cat Stevens
Greatest Hits	Dolly Parton
Sylvias Mother	Dr.Hook
…	…

XSLT <xsl:if> Element

To put a conditional if test against the content of the XML file, add an <xsl:if> element to the XSL document.

Syntax

```
<xsl:if test="expression">
        ... Some output if the expression is true...
</xsl:if>
```

Where to Put the <xsl:if> Element

To add a conditional test, add the <xsl:if> element inside the <xsl:for-each> element in the XSL file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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">
   <xsl:if test="price &gt; 10">
    <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="artist"/></td>
    </tr>
   </xsl:if>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

The value of the required test attribute contains the expression to be evaluated.

The code above will only output the title and artist elements of the CDs that has a price that is higher than 10.

The result of the transformation above will look like this:

My CD Collection

Title	**Artist**
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore
One night only	Bee Gees
Romanza	Andrea Bocelli
Black Angel	Savage Rose

XSLT <xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>
```

To insert a multiple conditional test against the XML file, add the <xsl:choose>, <xsl:when>, and <xsl:otherwise> elements to the XSL file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
    <xsl:choose>
     <xsl:when test="price &gt; 10">
      <td bgcolor="#ff00ff">
      <xsl:value-of select="artist"/></td>
     </xsl:when>
     <xsl:otherwise>
      <td><xsl:value-of select="artist"/></td>
     </xsl:otherwise>
    </xsl:choose>
   </tr>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```
The code above will add a pink background-color to the "Artist" column WHEN the price of the CD is higher than 10.

The result of the transformation will look like this:

My CD Collection

Title	**Artist**
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli

XSLT <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes. If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
```

```
</p>
</xsl:template>
<xsl:template match="title">
Title: <span style="color:#ff0000">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="artist">
Artist: <span style="color:#00ff00">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
</xsl:stylesheet>
```

The result of the transformation will look like this:

My CD Collection

Title: Empire Burlesque

Artist: Bob Dylan

Title: Hide your heart

Artist: Bonnie Tyler

Title: Greatest Hits

Artist: Dolly Parton

-WTS

XML DOM

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard. The DOM defines a standard for accessing documents like XML and HTML:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The DOM is separated into 3 different parts / levels:

- Core DOM - standard model for any structured document
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.

The XML DOM defines a standard way for accessing and manipulating XML documents. The DOM presents an XML document as a tree-structure. For anyone working with XML, knowing the XML DOM is a must.

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them. In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

In the DOM, everything in an XML document is a node.

DOM Nodes

According to the DOM, everything in an XML document is a **node**.

The DOM says:

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

DOM Example

Look at the following XML file (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" cover="paperback">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
 </book>
</bookstore>
```

The root node in the XML above is named <bookstore>. All other nodes in the document are contained within <bookstore>.

The root node <bookstore> holds four <book> nodes.

The first <book> node holds four nodes: <title>, <author>, <year>, and <price>, which contains one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00".

Text is Always Stored in Text Nodes

A common error in DOM processing is to expect an element node to contain text.

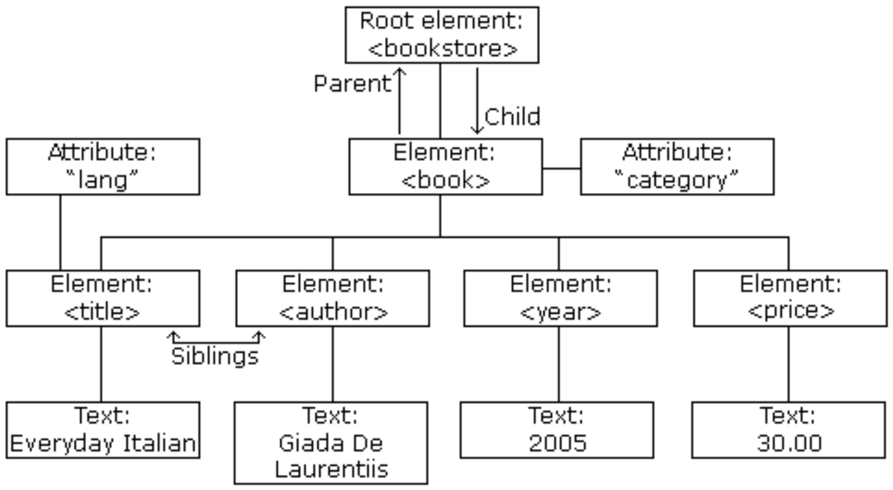
However, the text of an element node is stored in a text node. In this example: **<year>2005</year>**, the element node <year>, holds a text node with the value "2005". "2005" is **not** the value of the <year> element!

The XML DOM views an XML document as a node-tree. All the nodes in the tree have a relationship to each other.

The XML DOM Node Tree

The XML DOM views an XML document as a tree-structure. The tree structure is called a **node-tree.** All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.

The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:



The image above represents the XML file books.xml.

Node Parents, Children, and Siblings

The nodes in the node tree have a hierarchical relationship to each other.

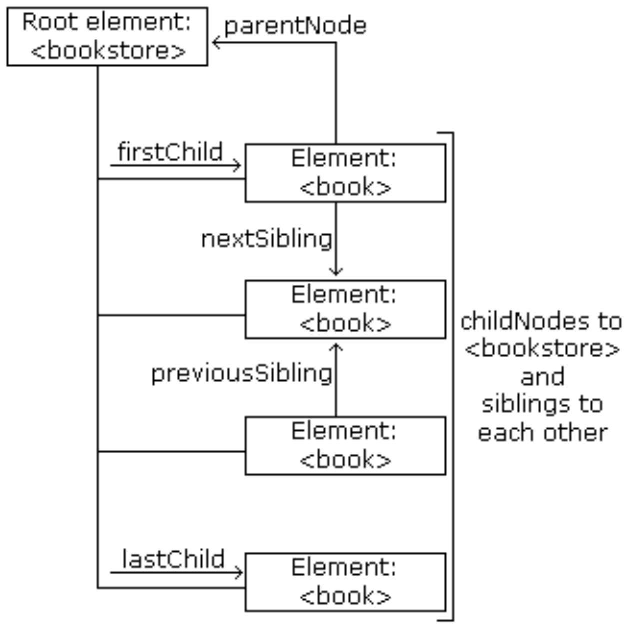
The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children

•

Siblings are nodes with the same parent

The following image illustrates a part of the node tree and the relationship between the nodes:



Because the XML data is structured in a tree form, it can be traversed without knowing the exact structure of the tree and without knowing the type of data contained within.

You will learn more about traversing the node tree in a later chapter of this tutorial.

First Child - Last Child

Look at the following XML fragment:

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

In the XML above, the <title> element is the first child of the <book> element, and the <price> element is the last child of the <book> element. Furthermore, the <book> element is the parent node of the <title>, <author>, <year>, and <price> elements.

[**DETAIL ON XML DOM** - http://goo.gl/eXCbG]

WHY XML Parser?

We need XML parser because we do not want to do everything in our application from scratch, and we need some "helper" programs or libraries to do something very low-level but very necessary to us. These low-level but necessary things include checking the well-formed-ness, validating the document against its DTD or schema (just for validating parsers), resolving character reference, understanding CDATA sections, and so on. XML parsers are just such "helper" programs and they will do all these jobs. With XML parsers, we are shielded from a lot of these complexities and we could concentrate ourselves on just programming at high-level through the API's implemented by the parsers, and thus gain programming efficiency.

SAX Parser

SAX is the Simple API for XML, originally a Java-only API. SAX was the first widely adopted API for XML in Java, and is a "de facto" standard. The current version is SAX 2.0.1, and there are versions for several programming language environments other than Java.

SAX (Simple API for XML) is an event-based sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

DOM parsers Vs. SAX parsers

- A DOM parser creates a tree structure in memory from the input document and then waits for requests from client. But a SAX parser does not create any internal structure. Instead, it takes the occurrences of components of an input document as events, and tells the client what it reads as it reads through the input document.
- A DOM parser always serves the client application with the entire document no matter how much is actually needed by the client. But a SAX parser serves the client application always only with pieces of the document at any given time.
- With DOM parser, method calls in client application have to be explicit and forms a kind of chain. But with SAX, some certain methods (usually overridden by the client) will be invoked automatically (implicitly) in a way, which is called "callback" when some certain events occur. These methods do not have to be called explicitly by the client, though we could call them explicitly.

In the following cases, using SAX parser is advantageous than using DOM parser.

- The input document is too big for available memory (actually in this case SAX is your only choice)
- You can process the document in small contiguous chunks of input.
- You do not need the entire document before you can do useful work
- You just want to use the parser to extract the information of interest, and all your computation will be completely based on the data structures created by yourself. Actually in most of our applications, we create data structures of our own which are usually not as complicated as the DOM tree.

In the following cases, using DOM parser is advantageous than using SAX parser.

- Your application needs to access widely separately parts of the document at the same time.
- Your application may probably use an internal data structure, which is almost as complicated as the document itself.
- Your application has to modify the document repeatedly.
- Your application has to store the document for a significant amount of time through many method calls.

XML processing with SAX

A parser that implements SAX (i.e., *a SAX Parser*) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include:

- XML Text nodes
- XML Element Starts and Ends
- XML Processing Instructions
- XML Comments

Some events correspond to XML objects that are easily returned all at once, such as comments. However, XML *elements* can contain many other XML objects, and so SAX represents them, as does XML itself: by one event at the beginning, and another at the end. Properly speaking, the SAX interface does not deal in *elements*, but in *events* that largely correspond to *tags*. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

There are many SAX-like implementations in existence. In practice, details vary, but the overall model is the same. For example, XML attributes are typically provided as name and value arguments passed to element events, but can also be provided as separate events, or via a hash or similar collection of all the attributes. For another, some implementations provide "Init" and "Fin" callbacks for the very start and end of parsing; others don't. The exact names for given event types also vary slightly between implementations.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
        <DocumentElement param="value">
                <FirstElement>&#xb6; Some Text</FirstElement>
                <?some_pi some_attr="some_value"?>
                <SecondElement param2="something">
                        Pre-Text <Inline>Inlined text</Inline> Post-text.
                </SecondElement>
        </DocumentElement>
```

This XML document, when passed through a SAX parser, will generate a sequence of events like the following:

- XML Element start, named *DocumentElement*, with an attribute *param* equal to "value"
- XML Element start, named *FirstElement*
- XML Text node, with data equal to "¶ Some Text" (note: certain white spaces can be changed)
- XML Element end, named *FirstElement*
- Processing Instruction event, with the target *some_pi* and data *some_attr="some_value"* (the content after the target is just text; however, it is very common to imitate the syntax of XML attributes, as in this example)
- XML Element start, named *SecondElement*, with an attribute *param2* equal to "something"
- XML Text node, with data equal to "Pre-Text"
- XML Element start, named *Inline*
- XML Text node, with data equal to "Inlined text"
- XML Element end, named *Inline*
- XML Text node, with data equal to "Post-text."
- XML Element end, named *SecondElement*
- XML Element end, named *DocumentElement*

Note that the first line of the sample above is the XML Declaration and not a processing instruction; as such it will not be reported as a processing instruction event (although some SAX implementations provide a separate event just for the XML declaration).

The result above may vary: the SAX specification deliberately states that a given section of text may be reported as multiple sequential text events. Many parsers, for example, return separate text events for numeric character references. Thus in the example above, a SAX parser may generate a different series of events, part of which might include:

- XML Element start, named *FirstElement*
- XML Text node, with data equal to "¶" (the Unicode character U+00b6)
- XML Text node, with data equal to " Some Text"
- XML Element end, named *FirstElement*

[SRC: Wiki, WTS and more…]

Chapter 04

Advanced Technology

Introduction

Web development is a broad term for the work involved in developing a web site for the Internet (World Wide Web) or an intranet (a private network). Web development can range from developing the simplest static single page of plain text to the most complex web-based Internet applications, electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers, may include web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.

Among web professionals, "web development" usually refers to the main non-design aspects of building web sites: writing markup and coding. While coding, the developer may choose to core programming or s/he may choose to use any of the frameworks. To solve a complicated problem in scientific mathematics, you can either take a paper to work out, or you can use a scientific calculator to solve it. Working out mathematics in a paper is like coding in Core PHP, tapping a scientific calculator is like coding in Framework.

Core PHP – Maths with Paper

- Best student can solve the problem in few steps. Accuracy level – 75% to 100%.
- Average student may or may not solve the problem, he will write down few more steps to solve the same problem. Accuracy level – 50% to 75%.
- Poor student cannot solve the problems. Still he will write down, lot and lot of steps to solve the problem. Accuracy level – 0% to 50%.

Framework – Scientific Calculator

- Every student can solve the problem with 100% accuracy, once they learnt how to use the calculator.
- The predefined formulas in the calculator will provide you accurate results faster for any problem.

Problem with Core PHP

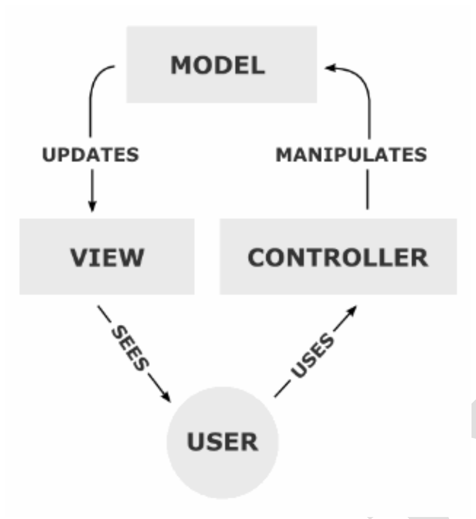
Core PHP becomes complicated, when people start writing their own logic in it. One can bring the output in few lines of code, where the other can take a few hundred of lines to bring the same. Both of them cannot read each other's code. So the problem starts here, that is inconsistency.

Is Core PHP that bad?

No, not at all. Core PHP helps you to understand the logics behind framework. Your logical thinking can be improved with Core PHP. The Core PHP becomes bad only when it goes to a bad programmer's desk. Don't dive into Framework without learning or coding in Core PHP. Make sure that you read the full documentation before you start coding in Framework, writing Core PHP inside Framework has become common nowadays, it's an insult to Frameworks. ☺

What is PHP Framework?

PHP is the world's most popular scripting language for many different reasons – flexibility, ease-of-use, among others – but often times coding in PHP, or any language for that matter, can get rather monotonous and repetitive. That's where a PHP framework can help. PHP frameworks streamline the development of web applications written in PHP by providing a basic structure for which to build the web applications. In other words, PHP frameworks help to promote rapid application development (RAD), which saves you time, helps build more stable applications, and reduces the amount of repetitive coding for developers. Frameworks can also help beginners to build more stable apps by ensuring proper database interaction and coding on the presentation layer. This allows you to spend more time creating the actual web application, instead of spending time writing repetitive code.



The general idea behind the workings of a PHP framework is referred to as Model View Controller (MVC). MVC is an architectural pattern in programming that isolates business logic from the UI, allowing one to be modified separately from the other (also known as separation of concerns). With MVC, Model refers to data; View refers to the presentation layer, and Controller to the application or business logic. Basically, MVC breaks up the development process of an application, so you can work on individual elements while others are unaffected. Essentially, this makes coding in PHP faster and less complicated.

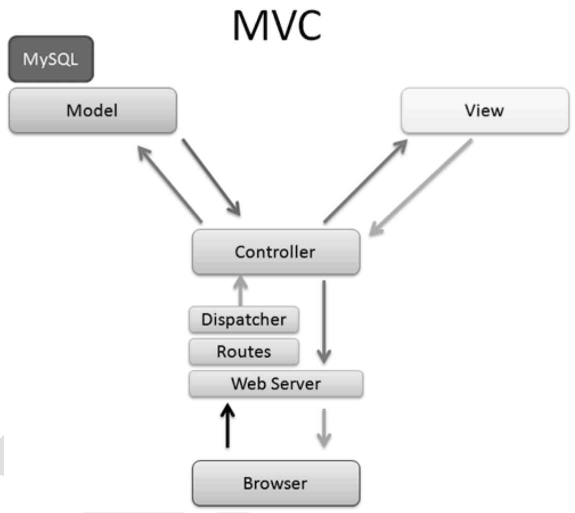
'The model-view-controller pattern was originally formulated in the late 1970s by Trygve Reenskaug at Xerox PARC, as part of the Smalltalk system.'

Model–view–controller (MVC) is a software architecture pattern that separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a pie chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view. The central ideas behind MVC are code reusability and separation of concerns.

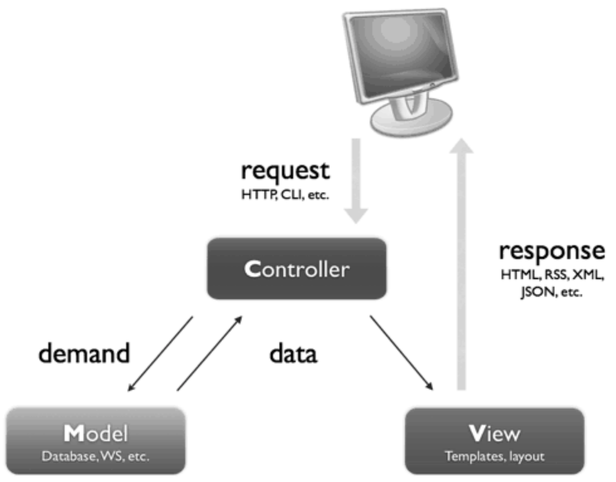
A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.

A **view** requests from the model the information that it needs to generate an output representation.

A **controller** can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).



Model View Controller has been adapted as architecture for World Wide Web applications. Several commercial and noncommercial application frameworks have been created that enforce the pattern. These frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server. Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server. As client technologies have matured, frameworks such as ***JavaScriptMVC*** and Backbone have been created that allow the MVC components to execute partly on the client (e.g. AJAX).



Why choose Framework?

Framework assures reliability, consistency and a big time-saver. It has rich set of functionalities, so you don't need to reinvent the wheel again and again. You will have almost all the functionalities to develop a PHP web application. Since it has been developed in OOPS, you can extend the existing functionality and create your own to have a full control over the application. Framework will not let you to write bad code, unless you purposely do it. When you work as a team, integrating your entire module becomes very easier; it also helps a lot in understanding each other's code. When you start developing a project, there are a lot of things, which you have to take care about, but we know only half of the things in it. Framework does everything for you, so you can be assured that your application is clean and safe. Inputs can be sanitized easily. MVC is one of the key functionality of Frameworks; separation of logics from views is a very good practice.

We all know that the client will surely come back to us one day to enhance the website with lot of his innovative requirements. If the project was done in Core PHP, you will have to say no for 50% of his new requirements, or simply you can tell him that the project is expired. But if the project was done is Framework, the beauty of Framework can be witnessed here. All you have to do is a cakewalk and give the updated project back to the client. You may have experienced it, if you are working in a web development company.

When working on a project with tight deadlines, utilizing a PHP framework is a huge benefit that can greatly speed up the coding process. So if you're in a time crunch, PHP frameworks can be very beneficial to you. Another instance when PHP frameworks should be a consideration is when you're working on projects with substantial amounts of monotonous coding, because it will help make the job much less tedious.

I REPEAT…

Developers should utilize PHP frameworks for various reasons, but the number one reason is for speeding up the development process. Reusing code across similar projects will save the developer a substantial amount of time and effort. A framework offers pre-built modules for performing tedious coding tasks, so the developer can spend their time on developing the actual application rather than re-building the foundation with each and every project.

Stability is another big reason developers are utilizing frameworks. While simplicity is one of PHP's greatest assets, and the reason many people prefer to use this scripting language, it can also be one of its biggest downfalls. It's fairly easy, especially for beginners, to write bad code and not even realize it. With PHP the application will often times still work, but unknowingly you may have opened up a large security hole in your coding that may be susceptible to attacks. It's important to remember that PHP is a very forgiving language, so it's even more important to make sure to tie up any loose ends in your coding – even if the application seems to be working properly.

Finally, the availability of PHP frameworks is extensive, and there are many different frameworks to choose from. You can even create your own, although many developers elect to choose from any of the most well-known frameworks due to their popularity, large support teams, and their forums/communities that allow you to interact with other developers who utilize the same framework. As a side note, you should always examine your project to first decide if you should even use a framework or not.

Some questions you should ask yourself are: Will it save you, and anyone else who may use it, time and effort? Will the app perform better? Will it improve stability? If you can answer yes to any of these questions, a PHP framework may be the right answer for that particular project.

What to Look for in a PHP Framework?

There are plenty of options available to anyone who may be searching for PHP frameworks, and there's even the option of creating your own, although that's only recommended for PHP experts. When searching for the PHP framework best suited for your needs, it's important to keep in mind who will be using and/or modifying your applications from top to bottom. If there are many people who will be using the application, it may be best use a popular PHP framework that many developers are familiar with. On the other hand, if you wish to build web applications for your own personal use, you are better off choosing any PHP framework that you're comfortable with – whether it's popular amongst the developer community or not.

Various factors to search for in a PHP framework includes: easy of use, rapid development/performance, and popularity amongst other developers, strong features, and support/forums. It's recommended to try out several PHP frameworks when you're first starting out in order to find one that suits your needs the best. All frameworks are slightly different and have varying strengths and weaknesses, for instance Zend Framework has been around since V3 and is full of features plus has an extensive support system in place since it has been around for so long. On the contrary, CakePHP is another PHP framework which is younger than Zend Framework and has slightly less of a support system in place but is much more user-friendly and easy to use.

Each type of PHP framework has its own advantages, so it's best to use a bit of trial and error to figure out which one will work the best for your needs. Another excellent way of choosing a framework is to consult the development community to see which ones they prefer. Those who have actually used a specific framework will be able to inform you of the ease-of-use, features, support availability, scope of the community surrounding the framework, and possible shortfalls.

Most Common Mistakes When Using a PHP Framework

Mistakes are possible in any type of programming, but PHP frameworks help to limit these mistakes greatly by providing good quality code that is tried and true from the start of the development process. Repetitive coding seems to promote mistakes now and then, and frameworks all but eliminate that problem.

That being said, there are still things to be careful of when utilizing any PHP framework. For instance, unless you are an expert in PHP programming, you should always opt for using a popular framework with plenty of support and an active user base. There are many frameworks out there that have little or no support, and/or individuals with limited knowledge of PHP created them. These types of frameworks can cause your applications to not function properly, and worse case scenario, could cause catastrophic security issues with your application.

Another somewhat common mistake is not ensuring your database and web server is compatible with the particular framework. If you don't meet its requirements, you won't be observing the best performance possible from your chosen framework. Even if you are an expert in PHP, you should always go over the documentation of the framework to confirm compatibility before trying it out.

Not following the recommended installation process of your PHP framework can also give you some headaches. The key is to take your time setting up the framework and follow the installation instructions to the “T” – The time you’ll save actually developing applications later will more than make up for the few extra minutes spent installing the framework correctly the first time.

Some of the Best PHP Frameworks Available

- **The Zend Framework**
    - The Zend Framework has a massive following amongst the development community and is focused on web 2.0 style applications. Because of their massive following, extensive support and active user base, Zend is referred to as “The PHP Company”. Zend is one of the most popular PHP frameworks available today. It has robust features that are built for corporate-level development, and it requires an extensive knowledge of PHP.
- **CakePHP**
    - CakePHP is a great choice for beginners to advanced PHP developers. It’s based on the same principles that Ruby on Rails is designed around, and it’s heavily focused on rapid development – making it a great framework to be used for rapid application development. Its rapidly growing support system, simplicity, and scalability make CakePHP one of the most popular PHP frameworks available today.
- **Symfony**
    - Symfony is aimed more at advanced developers who’s main objective is to create enterprise-level applications. This open source PHP framework is full of features and can do it all, but its main downfall is that it is a bit slower than other frameworks.
- **CodeIgniter**
    - CodeIgniter is well-known for its ease-of-use, performance and speed. Unlike Symfony, this PHP framework is ideal for shared hosting accounts or for when you want a framework with a small footprint. It offers simple solutions, and has an extensive library of video tutorials, forums, a user guide and wiki available for support. Beginners should consider using CodeIgniter.
- **Seagull**
    - Seagull is a well-established PHP framework used for building web, command line and GUI apps. It is an extremely easy to use framework that is ideal for beginners to advanced coders. For beginners, Seagull features a library of sample applications that can be customized to fit your needs, and for experts, Seagull offers a host of options – including best practices, standards, and modular codebase – for building web applications quickly and easily. Seagull has an active developer community and plenty of support documentation in place as well.

CodeIgniter

CodeIgniter is a powerful PHP framework with a very small footprint, built for PHP coders who need a simple and elegant toolkit to create full-featured web applications. If you're a developer who lives in the real world of shared hosting accounts and clients with deadlines, and if you're tired of ponderously large and thoroughly undocumented frameworks, then CodeIgniter might be a good fit.

CodeIgniter is an open source web application framework, for use in building dynamic web sites with PHP. "Its goal is to enable [developers] to develop projects much faster than...writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries."

The first public version of CodeIgniter was released on February 28, 2006, and the latest stable version 2.1.3 was released October 8, 2012.

CodeIgniter is loosely based on the popular Model-View-Controller development pattern. While view and controller classes are a necessary part of development under CodeIgniter, models are optional.

CodeIgniter is most often noted for its speed when compared to other PHP frameworks. In a critical take on PHP frameworks in general, PHP creator *Rasmus Lerdorf* also told that he liked CodeIgniter "because it is faster, lighter and the least like a framework."

CodeIgniter Is Right for You if...

- You want a framework with a small footprint.
- You need exceptional performance.
- You need clear, thorough documentation.
- You are not interested in large-scale monolithic libraries.
- You need broad compatibility with standard hosting.
- You prefer nearly zero configuration.
- You don't want to adhere to restrictive coding rules.
- You don't want to learn another template language.
- You prefer simple solutions to complexity.
- You want to spend more time away from the computer.

Homepage: http://ellislab.com/codeigniter

Javascript Framework (jQuery)

A JavaScript library is a library of pre-written JavaScript, which allows for easier development of JavaScript-based applications, especially for AJAX and other web-centric technologies.

With the expanded demands for JavaScript, an easier means for programmers to develop such dynamic interfaces was needed. Thus, JavaScript libraries such as Prototype, script.aculo.us, Ext Core, jsPHP, MooTools and jQuery and JavaScript widget libraries such as Ext JS, DHTMLX, and Dojo Toolkit were developed, allowing for developers to concentrate more upon more distinctive applications of Ajax.

JavaScript libraries allow for easier integration of JavaScript with other web development technologies, such as CSS, PHP, Ruby, and Java. Almost all JavaScript libraries are released under copyleft license to ensure license-free distribution, usage, and modification.

There are a few JavaScript frameworks/toolsets out there, such as:

- jQuery;
- Dojo;
- Prototype;
- YUI;
- MooTools;
- ExtJS;
- SmartClient;
    - etc.

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.

jQuery is a multi-browser JavaScript library designed to simplify the client-side scripting of HTML. It was released in January 2006 at BarCamp, NYC by ***John Resig***. It is currently developed by a team of developers led by ***Dave Methvin***. Used by over 55% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today.

jQuery is free, open source software, licensed under the MIT License. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications.

Features

- DOM element selections using the multi-browser open source selector engine Sizzle, a spin-off out of the jQuery project
- DOM traversal and modification (including support for CSS 1-3)
- DOM manipulation based on CSS selectors that uses node elements name and node elements attributes (id and class) as criteria to build selectors
- Events
- Effects and animations
- AJAX
- Extensibility through plug-ins
- Utilities - such as user agent information, feature detection
- Compatibility methods that are natively available in modern browsers but need fall backs for older ones - For example the inArray() and each() functions.
- Multi-browser (not to be confused with cross-browser) support.

Including the library

The jQuery library is a single JavaScript file, containing all of its common DOM, event, effects, and Ajax functions. It can be included within a web page by linking to a local copy, or to one of the many copies available from public servers.

```
<script type="text/javascript" src="jquery.js"></script>
```

The most popular and basic way to introduce a jQuery function is to use the .ready() function.

```
$(document).ready(function() {
// script goes here
});
```

or the shortcut

```
$(function() {
// script goes here
});
```

[Homepage: http://jquery.com]

DHTML with jQuery

What is DHTML?

Dynamic HTML is not really a new specification of HTML, but rather a new way of looking at and controlling the standard HTML codes and commands. When thinking of dynamic HTML, you need to remember the qualities of standard HTML, especially that once a page is loaded from the server, it will not change until another request comes to the server. Dynamic HTML gives you more control over the HTML elements and allows them to change at any time, without returning to the Web server.

There are four parts to DHTML:

- Document Object Model (DOM)
- Scripts
- Cascading Style Sheets (CSS)
- HTML

DOM

The DOM is what allows you to access any part of your Web page to change it with DHTML. The DOM specifies every part of a Web page and using its consistent naming conventions you can access them and change their properties.

Scripts

Scripts written in either JavaScript or VBScript.

Cascading Style Sheets (CSS)

CSS is used in DHTML to control the look and feel of the Web page. Style sheets define the colors and fonts of text, the background colors and images, and the placement of objects on the page. Using scripting and the DOM, you can change the style of various elements.

HTML

HTML is used to create the page itself and build the elements for the CSS and the DOM to work on. There is nothing special about HTML for DHTML - but having valid HTML is even more important.

Most common features of DHTML:

- **Changing the tags and properties**
  - This is one of the most common uses of DHTML. It allows changing the qualities of an HTML tag depending on an event outside of the browser (such as a mouse click, time, or date, and so on). With this information can be preloaded but not displayed until the reader clicks on a specific link.

- **Real-time positioning**
    - When most people think of DHTML this is what they expect. Objects, images, and text moving around the Web page. This can allow you to play interactive games with your readers or animate portions of your screen.

HTML5 is the new DHTML

{… … …}

Simple Ajax with jQuery

'AJAX stands for asynchronous JavaScript and XML.'

Ajax (AJAX) an acronym for Asynchronous JavaScript and XML is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the *XMLHttpRequest* object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.

Ajax is not a single technology, but a group of technologies. HTML and CSS can be used in combination to mark up and style information. The DOM is accessed with JavaScript to dynamically display, and to allow the user to interact with the information presented. JavaScript and the *XMLHttpRequest* object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

- In Gmail, switch from inbox to draft. Part of the page is changed, but the page is not refreshed. You remain on the same page. Url has not changed (except for the #draft at the end of the url, but that's still the same webpage).
- In Google Reader, select a feed. The content changes, but you are not redirected to another url.
- In Google Maps, zoom in or zoom out. The map has changed, but you remain on the same page.

The key to AJAX's concept is "asynchronous". This means something happens to the page after it's loaded. Traditionally, when a page is loaded, the content remains the same until the user leaves the page. With AJAX, JavaScript grabs new content from the server and makes changes to the current page. This all happens within the lifetime of the page, no refresh or redirection is needed.

With the jQuery AJAX methods, you can request text, HTML, XML, or JSON from a remote server using both HTTP Get and HTTP Post - And you can load the external data directly into the selected HTML elements of your web page!

The Old Ajax Way

```
var xmlhttp;
xmlhttp = GetXmlHttpObject();
if(xmlhttp == null){
  alert("OOoo! Your browser doesn't support AJAX!");
  return;
}
xmlhttp.onreadystatechange = stateChanged;
xmlhttp.open("GET", "http://www.google.com", true);
xmlhttp.send(null);

function stateChanged(){
  if(xmlhttp.readyState == 4){
    // do something with the response text
    alert(xmlhttp.responseText);
  }
}
function GetXmlHttpObject(){
  // IE7+, Firefox, Chrome, Opera, Safari
  if(window.XMLHttpRequest){
    return new XMLHttpRequest();
  }
  //IE5, IE6
  if(window.ActiveXObject){
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
  return null;
}
```

The code above simply request the contents of Google's home page, and then alert in the window. The code is too long.

Writing regular AJAX code can be a bit complicated, because different browsers have different syntax for AJAX implementation. You will have to write extra code to test for different browsers. However, the jQuery team has taken care of this, so that we can write AJAX functionality with only one single line of code.

Alternatively, here is the exact same function written in jQuery:

```
$.ajax({
  url: 'http://www.google.com',

  success:function(data){

    alert(data);

  }

});
```

The jQuery AJAX method is much easier. You give it a URL to request, and then you give it a set of instructions if a successful request is made. jQuery function handles 99% of all the AJAX requests we need to make, it includes a success and failure function to ensure that users get the proper feedback they need, and we'll also get a spinning loading image while the request is being processed to boot.

Some ways to make AJAX calls with jQuery:

- **load()**: Load a piece of html into a container DOM.
- **$.getJSON()**: Load a JSON with GET method.
- **$.getScript()**: Load a JavaScript.
- **$.get()**: Use this if you want to make a GET call and play extensively with the response.
- **$.post()**: Use this if you want to make a POST call and don't want to load the response to some container DOM.
- **$.ajax()**: Use this if you need to do something when XHR fails, or you need to specify ajax options (e.g. cache: true) on the fly.

Ajax vs. jQuery

AJAX is a very powerful tool to use but it can't be utilized with simple HTML since HTML doesn't allow the page to be changed after it has fully loaded. In order to use AJAX, you would need a client side scripting language that allows you to detect the actions of the user and modify elements on the page accordingly. jQuery does that exactly, that is why both are often used together to present web pages that a user can interact with easily without repetitive loading.

jQuery does all the work on the front end, therefore you would need to have a full understanding of it in order to properly set-up your page. You would not need to learn the exact mechanisms of AJAX in order to utilize it as jQuery gives you an AJAX command to retrieve whichever data you need from the server.

Although the use of jQuery and AJAX makes the browsing experience a lot better for the user, the effect to the server hosting these files are not as desirable. Every time you make another AJAX request, a new connection to the server is made. Too many connections can sometimes be difficult for the server to cope with.

Summary:

1. jQuery is a lightweight client side-scripting library while AJAX is a combination of technologies used to provide asynchronous data transfer.
2. jQuery and AJAX are often used in conjunction with each other.
3. jQuery is primarily used to modify data on the screen dynamically and it uses AJAX to retrieve data that it needs without changing the current state of the displayed page.
4. Heavy usage of AJAX functions often cause server overloads due to the greater number of connections made

[*READ MORE:* AJAX in jQuery - http://goo.gl/cH3ct / http://goo.gl/nbqKN]