# Protocols governing Web

1) HTTP

How web browsers and web servers communicate?

- Access, send and receive HTML files on the internet.
- Request and response protocol between clients and servers.

2) HTTP's

- Highly advanced and secured version of HTTP.
- Uses port no. 443 for data communication.
- Allows secure transaction by encrypting the entire communication with Secure Socket layer.

## Difference in HTTP and HTTP's

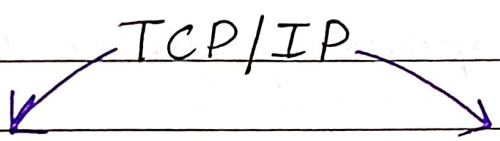
1) HTTP lacks security mechanism to encrypt the data whereas HTTP's provides SSL or TLS to secure communication b/w server and client.

2) HTTP operates on port 80 whereas HTTP's by default operates on port 443.

3) HTTP operates at the application layer whereas HTTP's by default operates on port 443.

4) HTTP transfer's data in plain text while HTTP's transfer data in cipher text or encrypted text.

5) HTTP is fast as compared to HTTP's




communication between computers on Internet

logical addressing protocols

→ ensures the delivery of information packets across network.
→ Connection oriented protocol.
→ UDP: User Datagram Protocol
→ FTP: File Transfer Protocol.

POP3: Post Office Protocol (Version 3)
SMTP: Simple Mail Transfer Protocol
SNMP: Simple Network Management Protocol.

Creating a good website

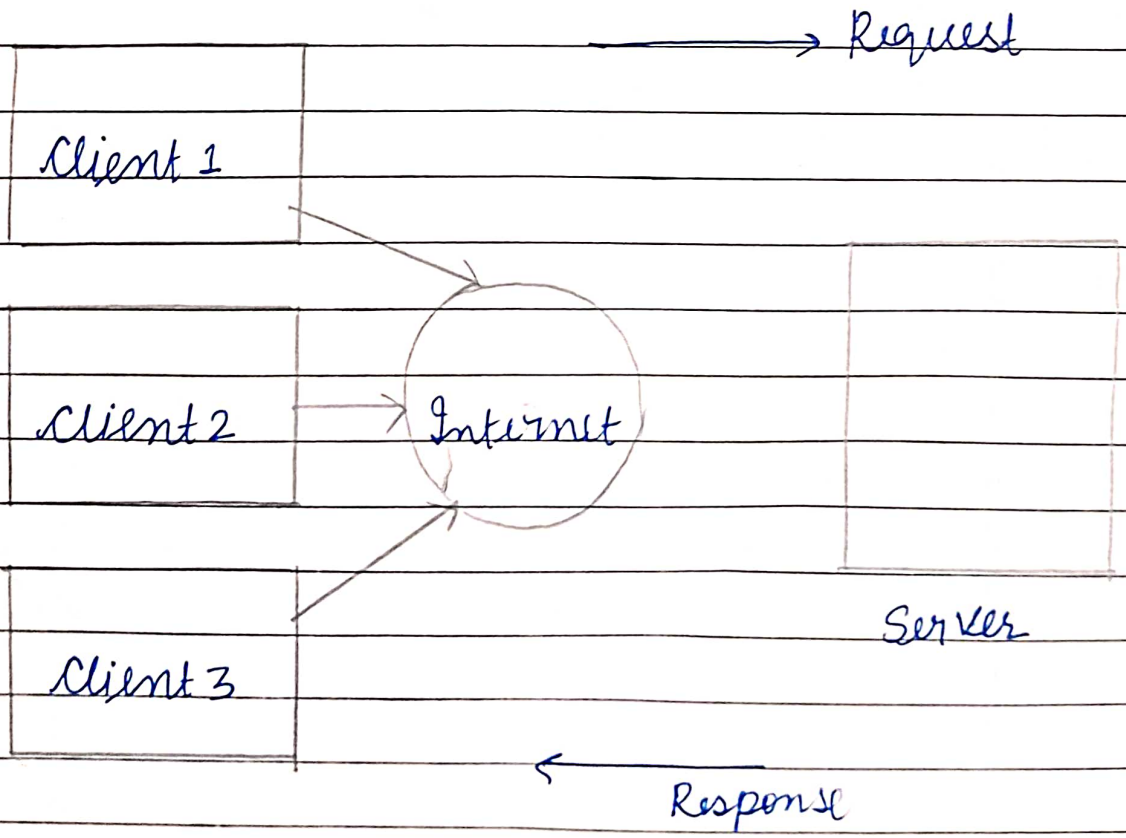
1) Build for your visitors, not for yourself.
2) Planning and preparing.
3) Find and build your theme.
4) Domain name and Brand Building.
5) Use a clean and simple web design.

6) Maintain small webpages.
7) Use widely accepted HTML format.
8) Write quality content for reader not yourself.
9) Write short paragraph and include header
10) Setup good navigation structure.

## Define your project

1) Write a Project Mission Statements.
2) Identify objectives
3) Identify your Target users.
   i) Market Research
   ii) Focus Groups
4) Determine the scope
5) Budget.

## Client Server Architecture



Date:
Page No.

# Client Server Application

## Client:

An application program running on local machine that requests for a service from an application program known as server program running on a remote machine.

→ runs only when it requests for a service from the server.
→ typically used for tasks such as displaying information entering data and controlling applications.

## Server:

Program that runs on remote machine providing services to the clients.

→ typically used for tasks such as storing data, processing data, and providing services to client computers.
→ runs all time as it does not known when it's service is required.

## Advantages

1) Centralised
2) Security
3) Performance
4) Scalability
5) Efficiency
6

Date:
Page No.

## Disadvantage

1) Complexity
2) Cost
3) Security

## HTML

### 1) Frames

```
<html>
<head>
        <title>          </title>
</head>
<body>
<frameset cols="30%, 40%, 30%">
<frame name="top" src="1.png" />
<frame name="main" src="2.png" />
<frame name="bottom" src="3.png" />
</frameset>
</body>
</html>
```

# XML - Extensible Markup Language

- Designed to store and transport data (to be read by people or by machines.
- Design to carry data, not to display data.
- Tags are not predefined need to define your own tags.
- Platform independent and language independent.
- Highly structured data embedded.
- Data stored in XML is known as being "self-defining"
- Used for data interchange b/w different systems, enabling the sharing of structured information.

## Difference between XML and HTML

| HTML | XML |
|---|---|
| 1) Hypertext Markup Language. | 1) Extensible Markup Language. |
| 2) It has predefined tags | 2) We can define own tags |
| 3) It focuses on how data should look | 3) It focuses |
| 4) HTML is not case sensitive. | 4) XML is case-sensitive |
| 5) closing tag for every tag is not required | 5) It is mandatory to close each tag |

| HTML | XML |
| --- | --- |
| 6) We can have improper nested tags. | 6) All elements must be properly nested within each other. |
| 7) It is static | 7) It is dynamic |

## Advantages of XML

1) Easy data exchange
2) Self describing data
3) Platform independent
4) RSS - Really Simple Syndication
   Really Simple Syndication

## Uses

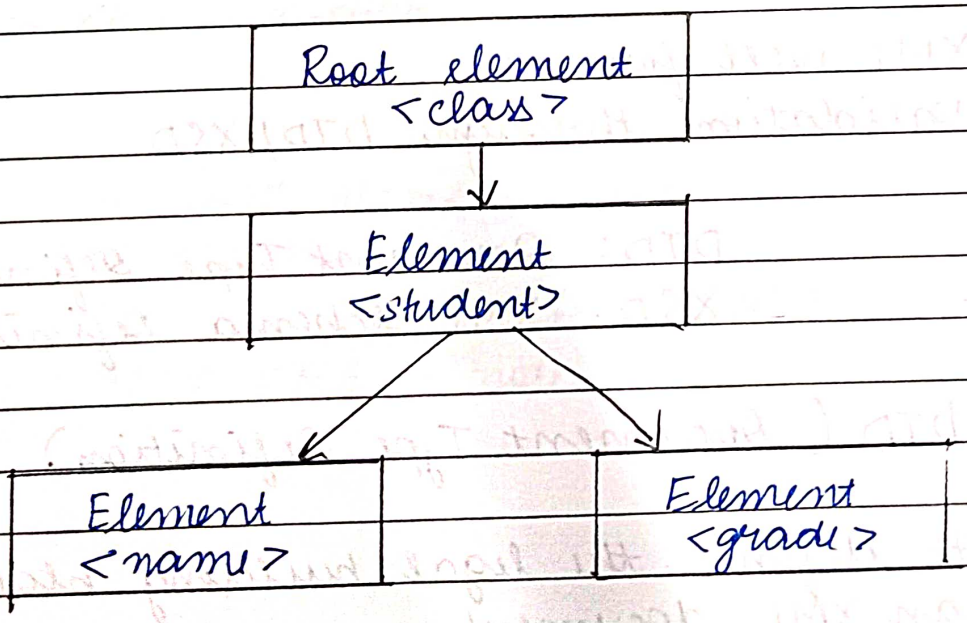
1) Data could be exchange over the web.
2) use of data from more resources and in more ways.
3) Used in RSS feeds. i.e delivering regularly changing web content.
   for eg:- news related sites
   cricbuzz

4) Used in web services.

example of XML:

```
<? xml version = "1.0" ? >
< class >        Root element
< student >      child
< name > XYZ < /name >   child
< grade > A < /grade >
< /student >
< student >
< name > PQR < /name >
< grade > A+ < /grade >
< /student >
< /class >
```

* XML Tree



# XML Syntax Rules

a) All XML elements must have closing tag.
b) XML tags are case sensitive.
c) All XML elements must be properly nested.
d) XML documents must have root element.
e) XML attribute value must be quoted.
f) Entity Refrence

"<"

if salary <1000 X
if salary <1000 ✓

<
>
&

g) Comments in XML

<!--- This is a Comment -->

## * XML Validation

a) XML well formed
b) Validation through DTD/XSD

DTD: Document Type Definition
XSD: XML Schema Definition.

### 1) DTD (Document Type Definition)

a) It defines the legal building blocks of an XML document.
b) It is used to define XML language precisely.

c) It is used to define structure of XML document.
d) It is used to perform validation.
e) It present XML syntax correctly or in validdd form.

Syntax:

```
<! DOCTYPE element DTD identifier
[ declaration 1
  declaration 2
]>
```

Types

Internal External

- within XML file
<! DOCTYPE root element
[ element declaration]>

- Outside XML file
<! DOCTYPE root element SYSTEM "filename">

example of DTD

```
<? xml version = "1.0" ? >
<! DOCTYPE note [
<! ELEMENT to (#PCDATA) >
<! ELEMENT note
```

Date:
Page No.

example of DTD (internal)

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>XYZ</to>
<from>PQR</from>
<heading>Reminder</heading>
<body>Learn HTML & CSS</body>
</note>
```

element name
attributes
datatypes

datatypes

#PCDATA
Parsed Character data

#CDATA
Character data

- if we use 123 in place of XYZ or PQR, it will give error because we use #PCDATA i.e use only character data.

Unique
Date:
Page No.

## example of DTD (external)

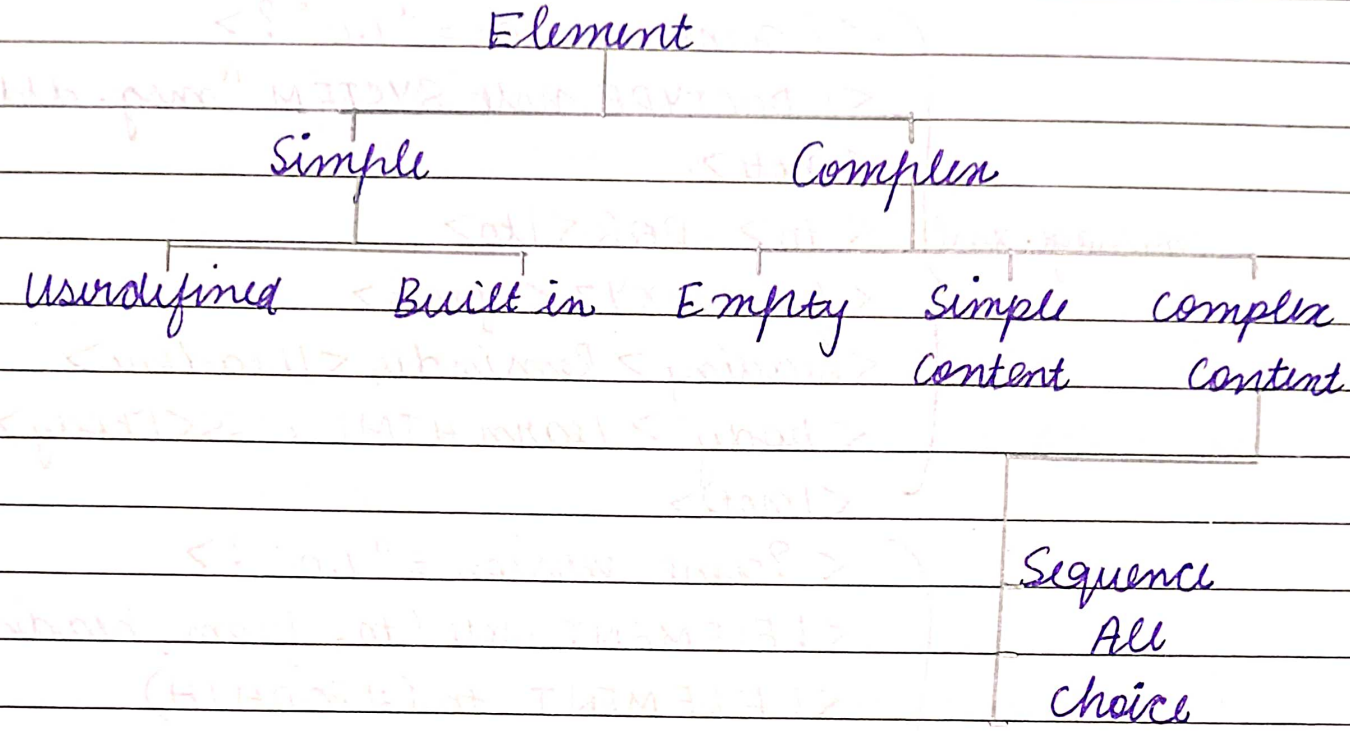
message.xml

```
<?xml version="1.0"?>
<!DOCTYDE note SYSTEM "msg.dtd">
<note>
<to> DQR </to>
<from> XYZ </from>
<heading> Reminder </heading>
<body> Learn HTML & CSS </body>
</note>
```

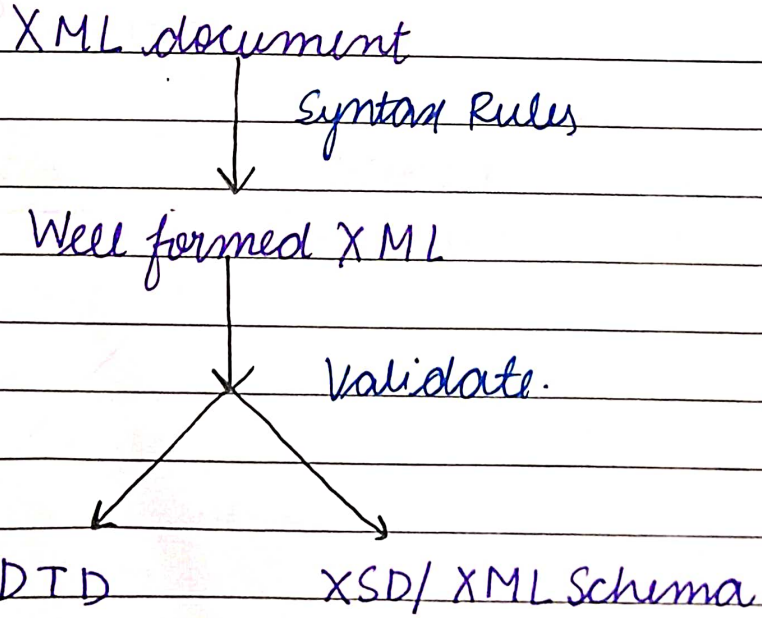
msg.dtd

```
<?xml version="1.0"?>
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)
<!ELEMENT from (#PCDATA)
<!ELEMENT heading (#PCDATA)
<!ELEMENT body (#PCDATA)
```

## XML Schema




## XML document



Ex: <!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
.
.
.
<!ELEMENT Body

## Limitation of XML document/DTD.

1) do not have built in datatypes.
2) do not support user defined data types.
3) doesn't define order for child elements.
4) allow only limited control over the number of occurences of an element within it's ~~format~~ parent.
5) do not support namespace or importing other schemas.

Ex: class.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:element name="class">
<xs:complexType>
<xs:sequence>
<xs:element name name="student">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs.string"/>
<xs:element name="grade" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

<?xml version="1.0" ?>
<class xsi:SchemaLocation = "class.xsd">
<student>
<name> PQR </name>
<grade> A </grade>
</student>
</class>

xsi → xschema instance

XML Parsers.

| | |
|---|---|
| It is used to read XML and create a way for programs to use XML | to access & modify the data in the doc. |

XML Doc → XML parser (DOM / SAX) → Client Application

Unique
Date:
Page No.

## DOM (Document Object Model)

- Tree based and reads an entire document.
- useful when reading small & medium size XML.
- defines a standard way to access & manipulate docs
- slow and consumes a lot of memory when it reads a large XML doc because it loads all the nodes into the memory for traversal and manipulation.

## SAX (Simple API for XML)

- it reads each unit of XML / node by node.
- high performance application or areas.
- fast and efficient to implement.
- consider to read a large size of XML doc.
- faster than DOM and use less memory.
- passes an XML file line by line.
- uses event driven serial across mechanism for accessing XML documents.

## Difference in DOM and SAX

| | SAX | DOM |
|---|---|---|
| 1) | Stands for Simple API for XML | 1) Stands for |
| 2) | It is event-based parser. | 2) Tree-Structured based parser. |

| | | |
|---|---|---|
| 3) | Runs little faster than DOM | 3) Runs little slower than SAX. |
| 4) | Best for larger size of XML. | 4) Best for small size |
| 5) | It is read only. | 5) It can insert or delete nodes. |
| 6) | Small part of XML file is only loaded in memory. | 6) It loads whole XML file in memory. |
| 7) | Top to bottom traversing. | 7) Traverse in any direction. |
| 8) | Less memory required | 8) Requires more memory. |
| 9) | It serves the client - application always with pieces of document at any given time. | 9) It always serves the client application with entire document, no matter how much is actually needed by client. |

Student.xml

```
<?xml version="1.0" ?>
<student-details>
    <Student>
        <Roll no> 1 </Roll no>
        <name> A </name>
        <sub> WT </sub>
        <marks> 90 </marks>
    </student>
        <Student>
        <Roll no> 2 </Roll no>
        <name> B </name>
        <sub> WT </sub>
        <marks> 95 </marks>
        </student>
</student-details>
```

parsesdemo.java

```
import org.wze.dom.*;
import javaxml.parses.*;
import java.util.*;
import java.io.*;
class parsesdemo {
        public static void main (String args [])
        {
Document BuilderFactory f = DocumentBuilderFactory.
                                newInstance();
```

Date:
Page No.

```
DocumentBuilder b = f.newDocumentBuilder();
Document d = b.parse("student.xml");
  Nodelist l = d.getElementByTagName("student");
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter roll no");
            int r = S.nextInt();
            int found=0;
for(int i=0; i<l.getlength(); i++){
        Node node = l.item(i);
if(node.getNodeType() == Node.ELEMENT_NODE)
{
   Element e = (Element) node;
     int x = Integer.paseInt(e.get
   ElementByTagName("Roll no").item(0)
                  get TextContent());
     if (x==r){
                  found=1;
     System.out.println("Student details");
     System.out.println("Student Rollno);
     System.out.println(e.getElementsByTagName
        ("Roll no"), item(0).getTextContent());
```

Objective of above code

Create an XML document that contain 10 student information, name, roll no, subject, marks. Write a program in Java which takes roll no as input and return student details by taking student info. from XML document using Parsing.