[Unit 4: HTTP and the Web Services]

Internet Technology (CSC-402)

Jagdish Bhatta

Central Department of Computer Science & Information Technology

Tribhuvan University

HTTP:

Refer Unit 3.

Web Servers and Web Access:

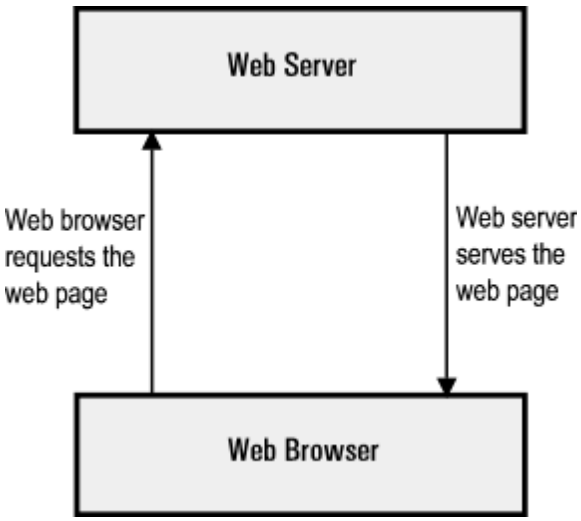
The primary function of a web server is to deliver web pages on the request to clients using the Hypertext Transfer Protocol (HTTP). This means delivery of HTML documents and any additional content that may be included by a document, such as images, style sheets and scripts.

A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary memory, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Web servers are not always used for serving the World Wide Web. They can also be found embedded in devices such as printers, routers, webcams and serving only a local network. The web server may then be used as a part of a system for monitoring and/or administering the device in question. This usually means that no additional software has to be installed on the client computer; since only a web browser is required (which now is included with most operating systems).

Whenever you view a web page on the internet, you are requesting that page from a web server. When you type a URL into your browser (for example, http://www.bhattag.com/internet_technology/index.cfm"), your browser requests the page from the web server and the web server sends the page back:



Basically to get service from the server:

1. **Your web browser first needs to know which IP address the website, say, "www.jagdish.com" resolves to. If it doesn't already have this information stored in it's cache, it requests the information from one or more DNS servers (via the internet). The DNS server tells the browser which IP address the website is located at. Note that the IP address was assigned when the website was first created on the web server.**
2. **Now that the web browser knows which IP address the website is located at, it can request the full URL from the web server.**
3. The web server responds by sending back the requested page. If the page doesn't exist (or another error occurs), it will send back the appropriate error message.
4. Your web browser receives the page and renders it as required.

When referring to web browsers and web servers in this manner, we usually refer to them as a *client* (web browser) and a *server* (web server).

There are many Web server software applications, including public domain software from NCSA and Apache, and commercial packages from Microsoft, Netscape and others.

Universal Naming with URI:

In the field of computer networking, a **URI scheme** is the top level of the uniform resource identifier (URI) naming structure. **All URIs and absolute URI references are formed with a scheme name, followed by a colon character (":"), and the remainder of the URI called *the scheme-specific part*.** The syntax and semantics of the scheme-specific part are left largely to the specifications governing individual schemes, subject to certain constraints such as reserved characters and how to "escape" them.

URI schemes are frequently and incorrectly referred to as "protocols", or specifically as **URI protocols** or **URL protocols**, since most were originally designed to be used with a particular protocol, and often have the same name. The http scheme, for instance, is generally used for interacting with Web resources using HyperText Transfer Protocol. Today, URIs with that scheme are also used for other purposes,

Every URI is defined as consisting of four parts, as follows:

<schema name>:<hierarchical part?[?<query>][#<fragment>]

The **scheme name** consists of a sequence of characters beginning with a letter and followed by any combination of letters, digits, plus ("+"), period ("."), or hyphen ("-"). Although schemes are case-insensitive, the canonical form is lowercase and documents that specify schemes must do so with lowercase letters. It is followed by a colon (":").

The **hierarchical part** of the URI is intended to hold identification information hierarchical in nature. Usually this part begins with a double forward slash ("//"), followed by an ***authority* part and an optional *path*.**

- The **authority** part holds an optional user-information part, terminated with "@" (e.g. username:password@); a hostname (e.g., domain name or IP address); and an optional port number, preceded by a colon ":".
- The **path** part is a sequence of segments (conceptually similar to directories, though not necessarily representing them) separated by a forward slash ("/"). Historically, each segment was specified to contain parameters separated from it using a semicolon (";"), though this was rarely used in practice and current specifications allow but no longer specify such semantics.

The **query** is an optional part, separated by a question mark ("?"), that contains additional identification information that is not hierarchical in nature. The query string syntax is not generically defined, but it is commonly organized as a sequence of <key>=<value> pairs, with the pairs separated by a semicolon or an ampersand. For example:

The **fragment** is an optional part separated from the front parts by a hash ("#"). It holds additional identifying information that provides direction to a secondary resource, e.g., a section heading (in an article) identified by the remainder of the URI. When the primary resource is an HTML document, the **fragment** is often an id attribute of a specific element and web browsers will make sure this element is visible.

Example

mailto:username@example.com?subject=Topic

ftp:// jagdish@ftp.example.org

Universal Naming Convention:

Universal Naning Convention (UNC) is the way to represent the path to a directory on a networked computer. It is of the form:

\\<name of network computer>\directory

i.e \\home\downloads

Although the UNC address looks similar to a URL the two are completely different. URLs use forward slashes rather than the backslash and the initial forward slashes are preceded by the transfer protocol (ftp, http) and a colon.

WWW Technology: HTML, DHTML, WML, XML:

HTML:

HTML stands for **hypertext markup language**. It is not a programming language. A markup language specifies the *layout and style* of a document. A markup language consists of a set of **markup tags**. HTML uses markup tags to describe web pages. HTML tags are keywords surrounded by **angle brackets** like <html>. Most HTML tags normally come in pairs like and . The first tag is called the **start tag** (or **opening tag**) and the second tag is called the **end tag** (or **closing tag**). HTML documents describe Web pages. HTML documents contain HTML tags and plain text. HTML documents are also called Web pages. A web browser read HTML documents and displays them as Web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. A simple HTML document is given below:

```
<html>
        <head>
                <title>This is my first web page</title>
        </head>
        <body>
                <h1>My first heading</h1>
                <p>My first paragraph</p>
        </body>
</html>
```

Save this page with **.html** or **.htm** extension. However, it is good practice to use **.htm** extension.

HTML Elements

HTML documents are defined by HTML elements. An HTML element is everything from the start tag to the end tag. For example, My first paragraph . An HTML element consists of start tag, end tag, and element content. The element content is everything between the start tag and end tag. Empty elements are closed in the start tag. Most HTML elements can have attributes. For example, **src** attribute of **img** tag.

HTML Attributes

Attributes provide additional information about HTML elements. Attributes are always specified in the start tag. Attributes come in name/value pair like name = "value". For example, HTML links are defined with **<a>** tag and the link address is provided as an attribute **href** like **cdcsit**.
Note: Always quote attribute values and use lowercase attributes.

HTML Headings
HTML headings are defined with the <h1> to <h6> tags. <h1> displays largest text and <h6> smallest. For example, <h1>My first heading</h1>.

HTML Paragraphs
HTML paragraphs are defined with tag. For example, My first paragraph .

HTML Rules (Lines)
We use <hr /> tag to create horizontal line.

HTML Comments
We use comments to make our HTML code more readable and understandable. Comments are ignored by the browser and are not displayed. Comments are written between <!-- and -->. For example, <!-- This is a comment -->.

HTML Line Breaks
If you want a new line (line break) without starting a new paragraph, use
 tag.

HTML Formatting Tags
We use different tags for formatting output. For example, <b> is used for bold and <i> is used for italic text. Some other tags are <big>, <small>, <sup>, <sub> etc.

HTML Styles
It is a new HTML attribute. It introduces CSS to HTML. The purpose of style attribute is to provide a common way to style all HTML elements. For example, <body style = "background-color:yellow">, , <h1 style = "text-align:center"> etc.

(Have a review from Web technology !!!)

DHTML:

DHTML stands for **D**ynamic **HTML**. **DHTML is the art of combining HTML, JavaScript, DOM, and CSS** and is NOT a language or a web standard. According to the World Wide Web Consortium (W3C): ***"Dynamic HTML is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated."***

The W3C HTML 4 standard has rich support for dynamic content:

- HTML supports JavaScript
- HTML supports the Document Object Model (**DOM**)
- HTML supports HTML Events
- HTML supports Cascading Style Sheets (**CSS**)

DHTML is about using these features, to create dynamic and interactive web pages. DHTML allows authors to add effects to their pages without the overhead of server-side programs or complicated sets of controls to achieve special effects.For example, DHTML allows the page author to:

- Animate text and images in their document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.
- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.
- Use a form to capture user input, and then process and respond to that data without having to send data back to the server.
- Include rollover buttons or drop-down menus.

The **term "DHTML" has fallen out of use in recent years as it was associated with practices and conventions that tended to not work well between various web browsers.** DHTML may now be referred to as **unobtrusive JavaScript coding** (DOM Scripting), in an effort to place an emphasis on agreed-upon best practices while allowing similar effects in an accessible, standards-compliant way. **The unobtrusive JavaScript includes separation of of functionality (the "behavior layer") from a Web page's structure/content and presentation.** Best practices to avoid the problems of traditional JavaScript programming. In traditional javascript we write as;

<input type="text" name="date" onchange="validateDate()" />

The unobtrusive solution is to register the necessary event handlers programmatically, rather than inline. Rather than adding the onchange attribute

explicitly as above, the relevant element(s) are simply identified, for example by class, id or some other means in the markup:
<input type="text" name="date" id="date" **/>**

A script that runs when the page is first loaded into the browser can then look for the relevant element(s) and set them up accordingly:

```
window.onload = function() {
   document.getElementById('date').onchange = validateDate;
};
```

Few examples;

1.

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.style.color='red'">Click Me!</h1>

</body>
</html>
```

2.

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

<script type="text/javascript">
function mymessage()
{
alert("This message was triggered from the onload event");
}
</script>
</head>

<body onload="mymessage()">
</body>

</html>
```

WML:

Wireless Markup Language (WML), based on XML, is a markup language intended for devices that implement the Wireless Application Protocol (WAP) specification, such as mobile phones. It provides navigational support, data input, hyperlinks, text and image presentation, and forms, much like HTML (HyperText Markup Language). It preceded the use of other markup languages now used with WAP, such as HTML itself, and XHTML (which are gaining in popularity as processing power in mobile devices increases).

WML Decks and Cards:

A main difference between HTML and WML is that the basic unit of navigation in HTML is a page, while that in WML is a card. A WML file can contain multiple cards and they form a deck.

When a WML page is accessed from a mobile phone, all the cards in the page are downloaded from the WAP server. So if the user goes to another card of the same deck, the mobile browser does not have to send any requests to the server since the file that contains the deck is already stored in the wireless device.

You can put links, text, images, input fields, option boxes and many other elements in a card.

A WML document is known as a "deck". Data in the deck is structured into one or more "cards" (pages) – each of which represents a single interaction with the user.

A WML program is typically divided into two parts: the document prolog and the body. Consider the following code:

Following is the basic structure of a WML program:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">

<wml>

    <card id="one" title="First Card">
    <p>
    This is the first card in the deck
```

```
</p>
</card>

<card id="two" title="Second Card">
<p>
Ths is the second card in the deck
</p>
</card>

</wml>
```

WML Document Prolog:

The first line of this text says that this is an XML document and the version is 1.0. The second line selects the document type and gives the URL of the document type definition (DTD). The DTD referenced is defined in WAP 1.2, but this header changes with the versions of the WML. The header must be copied exactly so that the tool kits automatically generate this prolog.

The prolog components are not WML elements and they should not be closed, i.e. you should not give them an end tag or finish them with />.

WML Document Body:

The body is enclosed within a <wml> </wml> tag pair. The body of a WML document can consist of one or more of the following:

- Deck
- Card
- Content to be shown
- Navigation instructions

Unlike HTML 4.01 Transitional, text cannot be enclosed directly in the <card>...</card> tag pair. So you need to put a content inside ... as shown above.

WBXML:

WAP Binary XML (**WBXML**) is a binary representation of XML. It was developed by the WAP Forum and is now maintained by the Open Mobile Alliance as a standard to allow XML documents to be transmitted in a compact manner over mobile networks and proposed as an addition to the World Wide Web Consortium's Wireless Application Protocolfamily of standards. The MIME media type application/vnd.wap.wbxml has been defined for documents that use WBXML.

WBXML is used by a number of mobile phones. Usage includes SyncML (Synchronization markup language) for transmitting address book and calendar data, Wireless Markup Language

WML Script:

WMLScript is the dialect of **JavaScript** used for WML pages and is part of the Wireless Application Protocol (WAP). WMLScript is a client-side scripting language and is very similar to JavaScript. Just like JavaScript WMLScript is used for tasks such as user input validation, generation of error message and other Dialog boxes etc.

WMLScript is based on **ECMAScript** (European Computer Manufacturers Association Script), which is JavaScript's standardized version. Thus the syntax of WMLScript is very similar to JavaScript but not fully compatible. A major difference between JavaScript and WMLScript is that JavaScript code can be embedded in the HTML markup, whereas WMLScript code is always placed in a file separate from the WML markup. URLs are used to refer to the actual WMLScript code in the WML document.

The following "Hello World" WMLScript example shows you how a WMLScript file typically looks like and demonstrates how to call WMLScript code in a WML document.

(helloWorldEg1.wml)

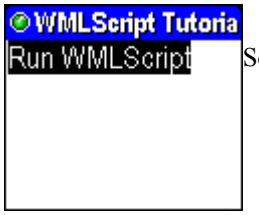
```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
"http://www.wapforum.org/DTD/wml13.dtd">

<wml>
  <card id="card1" title="WMLScript Tutorial">
    <p>
      <a href="helloWorldEg1.wmls#helloWorld()">Run WMLScript</a><br/>
      $(message)
    </p>
  </card>
</wml>
```

Here is the file that contains the WMLScript code:

(helloWorldEg1.wmls)

```
extern function helloWorld()
{
  WMLBrowser.setVar("message", "Hello World. Welcome to our WMLScript tutorial.");
  WMLBrowser.refresh();
}
```

Open the *helloWorldEg1.wml* file in a mobile phone browser and you can see something like this:



Sony Ericsson T68i

Nokia Mobile Browser 4.0

If you select the "Run WMLScript" link, the WMLScript function *helloWorld()* is executed and the line "Hello World. Welcome to our WMLScript tutorial." will appear in the mobile phone browser.

Sony Ericsson T68i

Nokia Mobile Browser 4.0

In the above example, the WMLScript code is not embedded in the WML markup and they are kept in separate files. This is the rule of WMLScript and you need to follow this when programming mobile Internet browsing applications.

There is only one function, *helloWorld()*, in the WMLScript file. The *extern* keyword is used to specify that the *helloWorld()* function is allowed to be called from outside the WMLScript file*helloWorldEg1.wmls*. The *extern* keyword is necessary here since we want to call the function from the WML file *helloWorldEg1.wml*.

Inside the *helloWorld()* function, we use two functions of the WMLBrowser standard library, *setVar()* and*refresh()*. The *setVar()* function is used to assign a value to a WML variable. We use the WMLScript code:

WMLBrowser.setVar("message", "Hello World. Welcome to our WMLScript tutorial.");

- to assign the value "Hello World. Welcome to our WMLScript tutorial." to a WML variable named*message*.

The *refresh()* function is used to instruct the WAP browser to refresh the current WML card. In the*helloWorld()* function, after we have assigned a value to the *message* variable, we make use of the line:

WMLBrowser.refresh();

- to refresh the WML card so that the change made to the *message* variable is shown on the screen of the mobile device.

To call the WMLScript function *helloWorld()* in the WML document, we use the URL below:

helloWorldEg1.wmls#helloWorld()

helloWorldEg1.wmls is the file that contains the WMLScript code and *helloWorld()* is the function to call.

XML:

XML is designed to transport and store data. XML stands for EXtensible Markup Language and is much like HTML. XML was designed to carry data, not to display data. XML tags are not predefined. You must define your own tags. XML is designed to be self-descriptive. **Extensible Markup Language** (**XML**) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML is not a replacement for HTML. XML and HTML were designed with different goals:

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

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

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information. The following example is a note to Tulsi, from Giri, stored as XML:

```
<note>
<to>Tulsi</to>
<from>Giri</from>
<heading>Reminder</heading>
```

<body>Don't forget to bunk web tech class at Patan!</body>
</note>

The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body. But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document. That is because the XML language has no predefined tags. However, the tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.). In contrast, XML allows the author to define his/her own tags and his/her own document structure. The XML processor can not tell us which elements and attributes are valid. As a result we need to define the XML markup we are using. To do this, we need to define the markup language's grammar. There are numerous "tools" that can be used to build an XML language – some relatively simple, some much more complex. They include DTD (Document Type Definition), RELAX, TREX, RELAX NG, XML Schema, Schmatron, etc.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.

XML Usages

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

XML Separates Data from HTML: If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML. With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

XML Simplifies Data Sharing: In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport: One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes: Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost. XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available: Different applications can access your data, not only in HTML pages, but also from XML data sources. With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

XML Used to Create New Internet Languages: A lot of new Internet languages are created with XML. Here are some examples:

- XHTML
- WSDL (Web Services Description Language) for describing available web services
- WAP and WML (Wireless Markup Language) as markup languages for handheld devices
- RSS (Really Simple Syndication / Rich Site Summary) languages for news feeds
- RDF (Resource Description Framework), a family of w3c spec, and OWL (Web Ontology Language) for describing resources and ontology
- SMIL (Synchronized Multimedia Integration Language) for describing multimedia for the web

XML Tree

XML documents form a tree structure that starts at "the root" and branches to "the leaves". XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
        <to>Tulsi</to>
        <from>Giri</from>
         <heading>Reminder</heading>
```

```
        <body>Don't forget to bunk the web tech class at Patan!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set). The next line describes the **root element** of the document (like saying: "this document is a note"):

<note>

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

<to>Tulsi</to>
<from>Giri</from>
<heading>Reminder</heading>
<body>Don't forget to bunk the web tech class at Patan!</body>

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

</note>

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

Thus, XML documents must contain a **root element**. This element is "the parent" of all other elements. The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements):

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

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters). All elements can have text content and attributes (just like in HTML).

XML Syntax Rules

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

1. **All XML Elements Must Have a Closing Tag**. In HTML, some elements may not have to have a closing tag, like;

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

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

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

2. **XML tags are case sensitive**. The tag <Letter> is different from the tag <letter>. Opening and closing tags must be written with the same case:

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

3. **XML Elements Must be Properly Nested**. In HTML, you might see improperly nested elements:

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

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

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

4. **XML Documents Must Have a Root Element**. XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

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

5. **XML Attribute Values Must be Quoted.** XML elements can have attributes in name/value pairs just like in HTML. In XML, the attribute values must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=06/01/2012>
  <to>Tulsi</to>
  <from>Giri</from>
</note>

<note date="06/01/2012">
  <to>Tulsi</to>
```

```
  <from>Giri</from>
</note>
```

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

6. **Entity Reference.** Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element. This will generate an XML error:

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

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

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

   There are 5 predefined entity references in XML:

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

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

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

8. White-space is preserved in XML**. HTML truncates multiple white-space characters to one single white-space:**

   HTML: Hello Tulsi
   Output: Hello Tulsi

   With XML, the white-space in a document is not truncated.

XML Elements

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

An element can contain:

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

Consider an example;

```
<bookstore>

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

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

</bookstore>
```

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

XML Elements are Extensible

XML elements can be extended to carry more information. Look at the following XML example:

```
<note>
<to>Tulsi</to>
<from>Giri</from>
<body>Don't forget to bunk the web tech class at Patan!</body>
</note>
```

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

MESSAGE

To: Tulsi
From: Giri

Don't forget to bunk the web tech class at Patan!

Suppose the XML document has been modified by adding some extra information to it like:

```
<note>
<date>2012-01-06</date>
<to>Tulsi</to>
<from>Giri</from>
<heading>Reminder</heading>
<body>Don't forget to bunk thee web tech class at Patan!</body>
</note>
```

Should the application break or crash?

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output. Thus, one of the beauties of XML, is that it can be extended without breaking applications.

(Have a review from Web Tech !!!)

WYSWIG Authoring Tool:

The somehow cryptic abbreviation WYSIWYG stands for "What You See Is What You Get". In such editors you edit not directly the source code of your documents, but its **presentation** as it (hopefully) will appear in the final document. So instead of writing blocks of code manually (as you e.g. would do it in Word or Latex), you manipulate with design components using an editor window. This means that you view something very similar to the end result while the document or image is being created. WYSIWYG (What You See Is What You Get) code generators offer speed and ease of use. Many of these editors do not require any knowledge of the programming languages generated by the software.

Some of these editors store pages in a proprietary format and then export them as HTML (possibly along with other formats); the user would continue to maintain the website by working with the files in the proprietary format and re-exporting them. Other, generally simpler WYSIWYG editors are designed to work directly with HTML files. Exported files tend to be larger than hand-coded pages (those produced with a text-based HTML editor or a plain text editor). WYSIWYG generators tend to be better than word processors at producing highly graphical and interactive pages. Some of the WYSIWYG tools are

- ASP.NET Web Matrix
- Adobe Dreamweaver (formerly Macromedia Dreamweaver)
- Amaya

- Microsoft Visual Studio
- Microsoft Visual Web Developer Express

A given HTML document will have an inconsistent appearance on various platforms and computers for several reasons:

Different browsers and applications will render the same markup differently: The same page may display slightly differently in Internet Explorer and Firefox on a high-resolution screen, but it will look very different in the perfectly valid text-only Lynx browser. It needs to be rendered differently again on a PDA, an internet-enabled television and on a mobile phone. Usability in a speech or braille browser, or via a screen-reader working with a conventional browser, will place demands on entirely different aspects of the underlying HTML. Printing the page, via different browsers and different printers onto various paper sizes, around the world, places other demands. With the correct use of modern HTML and CSS there is no longer any need to provide 'Printable page' links and then have to maintain two versions of the whole site. Nor is there any excuse for pages not fitting the user's preferred paper size and orientation, or wasting ink printing solid background colours unnecessarily, or wasting paper reproducing navigation panels that will be entirely useless once printed out.

Browsers and computer graphics systems have a range of user settings: Resolution, font size, colour, contrast etc can all be adjusted at the user's discretion, and many modern browsers allow even more user control over page appearance. All an author can do is suggest an appearance.

Web browsers, like all computer software, have bugs: They may not conform to current standards. It is hopeless to try to design Web pages around all of the common browsers' current bugs: each time a new version of each browser comes out, a significant proportion of the World Wide Web would need re-coding to suit the new bugs and the new fixes. It is generally considered much wiser to design to standards, staying away from 'bleeding edge' features until they settle down, and then wait for the browser developers to catch up to your pages, rather than the other way round.[9] For instance, no one can argue that CSS is still 'cutting edge' as there is now widespread support available in common browsers for all the major features,[10] even if many WYSIWYG and other editors have not yet entirely caught up.[11]

A single visual style can represent multiple semantic meanings: Semantic meaning, derived from the underlying structure of the HTML document, is important for search engines and also for various accessibility tools. On paper we can tell from context and experience whether bold text represents a title, or emphasis, or something else. But it is very difficult to convey this distinction in a WYSIWYG editor. Simply making a piece of text bold in a WYSIWYG editor is not sufficient to tell the editor *why* the text is bold - what the boldness represents semantically. What you see may be what most visitors get, but it is not guaranteed to be what *everyone* gets.

WYSIWYM (what you see is what you mean) is an alternative paradigm to WYSIWYG, in which the focus is on the semantic structure of the document rather than on the presentation. These editors produce more logically structured markup than is typical of WYSIWYG editors, while retaining the advantage in ease of use over hand-coding using a text editor. WYMeditor is such editor.

Helper applications:

CGI: Abbreviation of ***C****ommon* ***G****ateway* ***I****nterface*, a specification for transferring information between a World Wide Web server and a CGI program. A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic. CGI programs are the most common way for Web servers to interact dynamically with users. Many HTML pages that contain forms, for example, use a CGI program to process the form's data once it's submitted. Another increasingly common way to provide dynamic feedback for Web users is to include scripts or programs that run on the user's machine rather than the Web server. These programs can be Java applets, Java scripts, or ActiveX controls. These technologies are known collectively as *client-side* solutions, while the use of CGI is a *server-side* solution because the processing occurs on the Web server.

One problem with CGI is that each time a CGI script is executed, a new process is started. For busy Web sites, this can slow down the server noticeably. A more efficient solution, but one that it is also more difficult to implement, is to use the server's API, such as ISAPI or NSAPI. Another increasingly popular solution is to use Java servlets.

CGI is the Common Gateway Interface, a standard for programs to interface with information servers such as HTTP (web) servers. CGI allows the HTTP server to run an executable program or script in response to a user request, and generate output on the fly. This allows web developers to create dynamic and interactive web pages. Perl is a very common language for CGI programming as it is largely platform independent and the language's features make it very easy to write powerful applications.

It is important to remember that CGI is not a language in itself. CGI is merely a type of program which can be written in any language.

Perl:

Perl is a high-level, general-purpose, interpreted, dynamic programming language. Though Perl is not officially an acronym, there are various backronyms in usage, such as: *Practical Extraction and Reporting Language*. Perl was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing

easier. Since then, it has undergone many changes and revisions and become widely popular amongst programmers.

Larry Wall continues to oversee development of the core language, and its upcoming version, Perl 6. Perl borrows features from other programming languages including C, shell scripting (sh), AWK, and sed. The language provides powerful text processing facilities without the arbitrary data length limits of many contemporary Unix tools, facilitating easy manipulation of text files. Perl gained widespread popularity in the late 1990s as a CGI scripting language, in part due to its parsing abilities.

In addition to CGI, Perl is used for graphics programming, system administration, network programming, finance, bioinformatics, and other applications. Its major features include support for multiple programming paradigms (procedural, object-oriented, and functional styles), reference counting memory management (without a cycle-detecting garbage collector), built-in support for text processing, and a large collection of third-party modules.

Simple Perl Statements

```
print "Hello, world!\n"
say "Hello, world!"

$n = '3 apples';
$m = '2 oranges';
print $n + $m;
```

PHP: Will provide separate handout for PHP.

Note: *Java, JavaScript, Asp, .Net Appications* (Self Study… !!!)

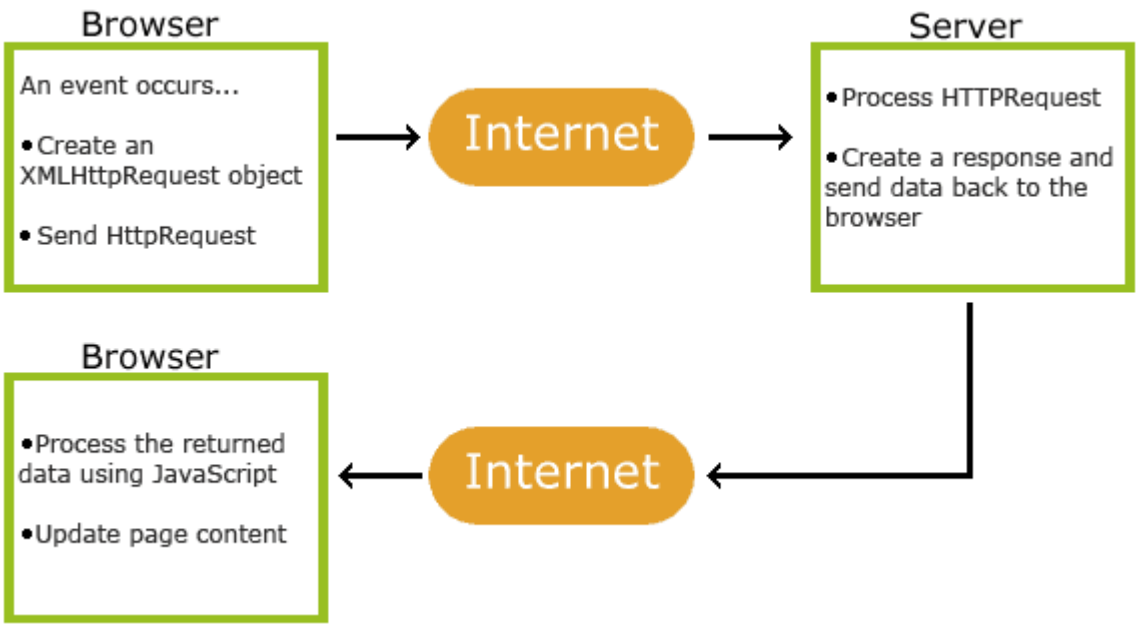
Introduction to Ajax:

Ajax (**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, **JSON** (or **JavaScript Object Notation**, is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for many languages), 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.

n the 1990s, most web sites were based on complete HTML pages; each user action required that the page be re-loaded from the server (or a new page loaded). This process is inefficient, as reflected by the user experience: all page content disappears then reappears, etc. Each time a page is reloaded due to a partial change, all of the content must be re-sent instead of only the changed information. This can place additional load on the server and use excessive bandwidth. Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works



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

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser and platform-independent. The keystone of AJAX is the XMLHttpRequest object.

Steps of AJAX Operation

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an asynchronous request to the Webserver.
5. Webserver returns the result containing XML document.
6. The XMLHttpRequest object calls the callback() function and processes the result.
7. The HTML DOM is updated

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object (IE5 and IE6 use an ActiveXObject). The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

***variable*=new XMLHttpRequest();**

Old versions of Internet Explorer (IE5 and IE6) uses an ActiveX Object:

***variable*=new ActiveXObject("Microsoft.XMLHTTP");**

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

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

Send a Request to a Server:

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

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

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. *method*: the type of request: GET or POST *url*: the location of the file on the server *async*: true (asynchronous) or false (synchronous)
send(*string*)	Sends the request off to the server. *string*: Only used for POST requests

GET or POST?

GET is simpler and faster than POST, and can be used in most cases. However, always use POST requests when:

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

GET Requests

A simple GET request:

If you want to send information with the GET method, add the information to the URL:

```
xmlhttp.open("GET","demo_get2.asp?fname=Henry&lname=Ford",true);
xmlhttp.send();
```

The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server:

```
xmlhttp.open("GET","ajax_test.asp",true);
```

The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Asynchronous - True or False?

AJAX stands for Asynchronous JavaScript and XML, and for the XMLHttpRequest object to behave as AJAX, the async parameter of the open() method has to be set to true:

xmlhttp.open("GET","ajax_test.asp",true);

Sending asynchronous requests is a huge improvement for web developers. Many of the tasks performed on the server are very time consuming. Before AJAX, this operation could cause the application to hang or stop.

With AJAX, the JavaScript does not have to wait for the server response, but can instead:

- execute other scripts while waiting for server response
- deal with the response when the response ready

Async=true

When using async=true, specify a function to execute when the response is ready in the onreadystatechange event:

Example

```
xmlhttp.onreadystatechange=function()
  {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
  }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Async=false

To use async=false, change the third parameter in the open() method to false:

xmlhttp.open("GET","ajax_info.txt",false);

Using async=false is not recommended, but for a few small requests this can be ok.

Remember that the JavaScript will NOT continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.

Note: When you use async=false, do NOT write an onreadystatechange function - just put the code after the send() statement:

Example

```
xmlhttp.open("GET","ajax_info.txt",false);
xmlhttp.send();
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

Server Response

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

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

The responseText Property

If the response from the server is not XML, use the responseText property. The responseText property returns the response as a string, and you can use it accordingly:

Example

```
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

The responseXML Property

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

Request the file cd_catalog.xml and parse the response:

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

The onreadystatechange event

When a request to a server is sent, we want to perform some actions based on the response. The onreadystatechange event is triggered every time the readyState changes. The readyState property holds the status of the XMLHttpRequest. Three important properties of the XMLHttpRequest object:

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

In the onreadystatechange event, we specify what will happen when the server response is ready to be processed.

When readyState is 4 and status is 200, the response is ready:

Example

```
xmlhttp.onreadystatechange=function()
  {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
 }
```

Note: The onreadystatechange event is triggered four times, one time for each change in readyState.

Using a Callback Function:

A callback function is a function passed as a parameter to another function. If you have more than one AJAX task on your website, you should create ONE standard function for creating the XMLHttpRequest object, and call this for each AJAX task.

The function call should contain the URL and what to do on onreadystatechange (which is probably different for each call):

Example

```
function myFunction()
{
loadXMLDoc("ajax_info.txt",function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
  });
}
```

Ajax framework:

In web application development, an **Ajax framework** is a framework which leverages Ajax, a collection of technologies for building dynamic web pages on the client side. While data is read from and sent to the server by JavaScript requests, frameworks may include server-side or client-side components to process the client's requests.

Some of the frameworks are JavaScript compilers, for generating JavaScript and Ajax that runs in the web browser client; some are pure JavaScript libraries; others are server-side frameworks that typically utilize JavaScript libraries. For examples; jquery Moo Tools etc.

Some Examples

```
<!DOCTYPE html>
<html>
<head>
        <script type="text/javascript">
        function loadXMLDoc()
        {
        var xmlhttp;
        if (window.XMLHttpRequest)
          {// code for IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp=new XMLHttpRequest();
          }
        else
          {// code for IE6, IE5
          xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
          }
```

```
        xmlhttp.onreadystatechange=function()
          {
          if (xmlhttp.readyState==4 && xmlhttp.status==200)
            {
            document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
            }
          }
        xmlhttp.open("GET","ajax_info.txt",true);
        xmlhttp.send();
        }
        </script>
</head>
<body>
        <div id="myDiv"><h2>Let AJAX change this text</h2></div>
        <button type="button" onclick="loadXMLDoc()">Change Content</button>
</body>
</html>
```

One more example;

It consist of a textbox. When a user types a character in the input field above, the function "showHint()" is executed. The function is triggered by the "onkeyup" event:

```
        function showHint(str)
        {
        var xmlhttp;
        if (str.length==0)
          {
          document.getElementById("txtHint").innerHTML="";
          return;
          }
        if (window.XMLHttpRequest)
          {// code for IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp=new XMLHttpRequest();
          }
        else
          {// code for IE6, IE5
          xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
          }
        xmlhttp.onreadystatechange=function()
          {
          if (xmlhttp.readyState==4 && xmlhttp.status==200)
            {
            document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
            }
          }
```

```
xmlhttp.open("GET","gethint.asp?q="+str,true);
xmlhttp.send();
}
```

If the input field is empty (str.length==0), the function clears the content of the txtHint placeholder and exits the function.

If the input field is not empty, the showHint() function executes the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the input field)

The page on the server called by the JavaScript above is an ASP file called "gethint.asp". The file is as;

```
<%
response.expires=-1
dim a(10)
'Fill up array with names
a(1)="Anna"
a(2)="Brittany"
a(3)="Cinderella"
a(4)="Diana"
a(5)="Eva"
a(6)="Fiona"
a(7)="Gunda"
a(8)="Hege"
a(9)="Inga"
a(10)="Johanna"

q=ucase(request.querystring("q"))       'get the q parameter from URL

if len(q)>0 then                        'lookup all hints from array if length of q>0
  hint=""
  for i=1 to 10
    if q=ucase(mid(a(i),1,len(q))) then
      if hint="" then
        hint=a(i)
      else
        hint=hint & " , " & a(i)
      end if
    end if
  next
end if
```

```
        'Output "no suggestion" if no hint were found or output the correct values
if hint="" then
  response.write("no suggestion")
else
  response.write(hint)
end if
%>
```

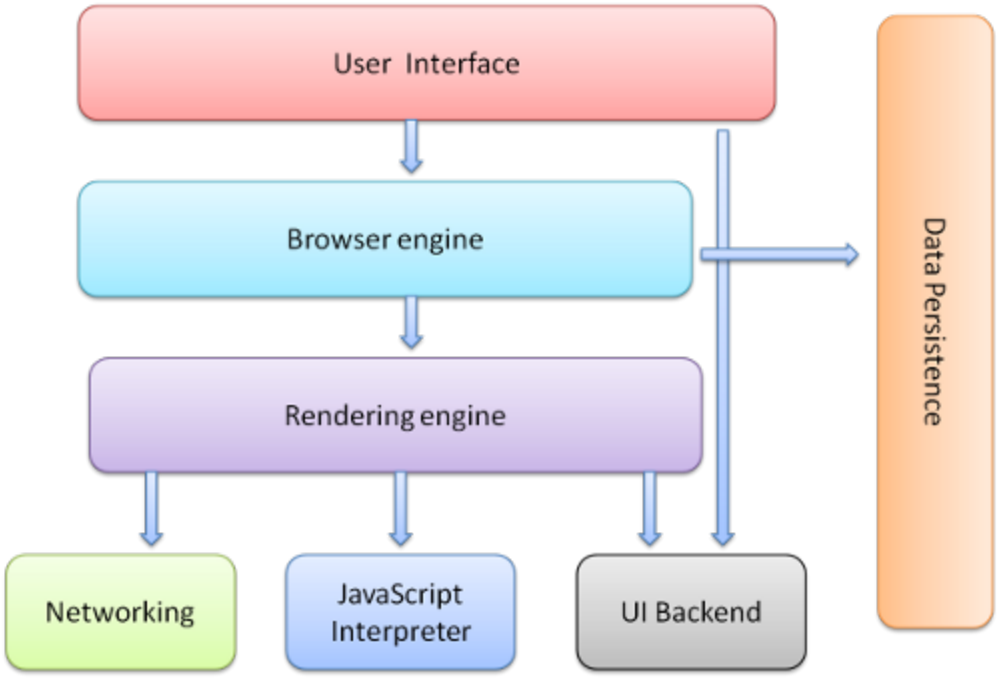
Browser as a rendering engine:

A web browser engine, (sometimes called layout engine or rendering engine), is a software component that takes marked upcontent (such as HTML, XML, image files, etc.) and formatting information (such as CSS, XSL, etc.) and displays the formatted content on the screen. It "paints" on the content area of a window, which is displayed on a monitor or a printer. **A web browser engine is typically embedded in web browsers, e-mail clients, on-line help systems or other applications that require the displaying (and editing) of web content. Engines may wait for all data to be received before rendering a page, or may begin rendering before all data is received.** This can result in pages changing as more data is received, such as images being filled in or a flash of unstyled content if rendering begins before formatting information is received.

The browser's main components are:

1. The user interface - this includes the address bar, back/forward button, bookmarking menu etc. Every part of the browser display except the main window where you see the requested page.
2. **The browser engine - the interface for querying and manipulating the rendering engine.**
3. **The rendering engine - responsible for displaying the requested content. For example if the requested content is HTML, it is responsible for parsing the HTML and CSS and displaying the parsed content on the screen.**
4. Networking - used for network calls, like HTTP requests. It has platform independent interface and underneath implementations for each platform.
5. UI backend - used for drawing basic widgets like combo boxes and windows. It exposes a generic interface that is not platform specific. Underneath it uses the operating system user interface methods.
6. JavaScript interpreter. Used to parse and execute the JavaScript code.

7. Data storage. This is a persistence layer. The browser needs to save all sorts of data on the hard disk, for examples, cookies. The new HTML specification (HTML5) defines 'web database' which is a complete (although light) database in the browser.



Browser main components.

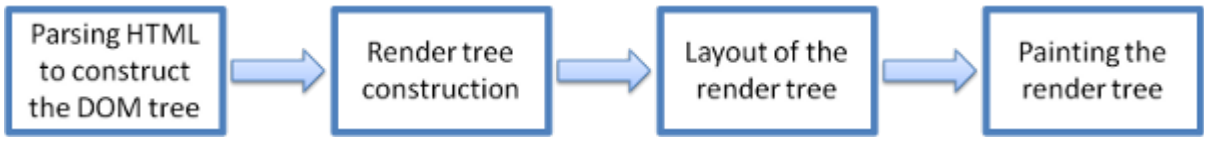
It is important to note that Chrome, unlike most browsers, holds multiple instances of the rendering engine - one for each tab, Each tab is a separate process.

The rendering engine:

The responsibility of the rendering engine, rendering, that is display of the requested contents on the browser screen. By default the rendering engine can display HTML and XML documents and images.

Our reference browsers - Firefox, Chrome and Safari are built upon two rendering engines. Firefox uses Gecko - a "home made" Mozilla rendering engine. Both Safari and Chrome use Webkit. Webkit is an open source rendering engine which started as an engine for the Linux platform and was modified by Apple to support Mac and Windows.

The rendering engine will start getting the contents of the requested document from the networking layer. This will usually be done in 8K chunks. After that this is the basic flow of the rendering engine:

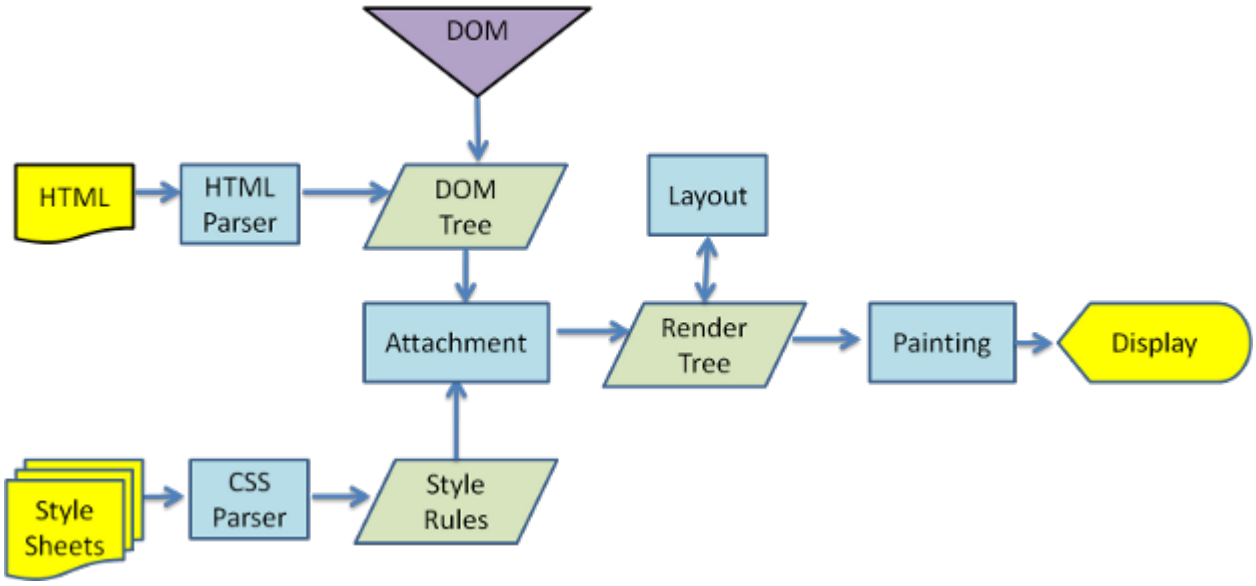


Rendering engine basic flow.

The rendering engine will start parsing the HTML document and turn the tags to DOM nodes in a tree called the "content tree". It will parse the style data, both in external CSS files and in style elements. The styling information together with visual instructions in the HTML will be used to create another tree - the render tree.

The render tree contains rectangles with visual attributes like color and dimensions. The rectangles are in the right order to be displayed on the screen. After the construction of the render tree it goes through a "layout" process. This means giving each node the exact coordinates where it should appear on the screen. The next stage is painting - the render tree will be traversed and each node will be painted using the UI backend layer.

It's important to understand that this is a gradual process. For better user experience, the rendering engine will try to display contents on the screen as soon as possible. It will not wait until all HTML is parsed before starting to build and layout the render tree. Parts of the content will be parsed and displayed, while the process continues with the rest of the contents that keeps coming from the network.



Render tree construction

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the

visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order.

Firefox calls the elements in the render tree "frames". Webkit uses the term renderer or render object. A renderer knows how to layout and paint itself and it's children.