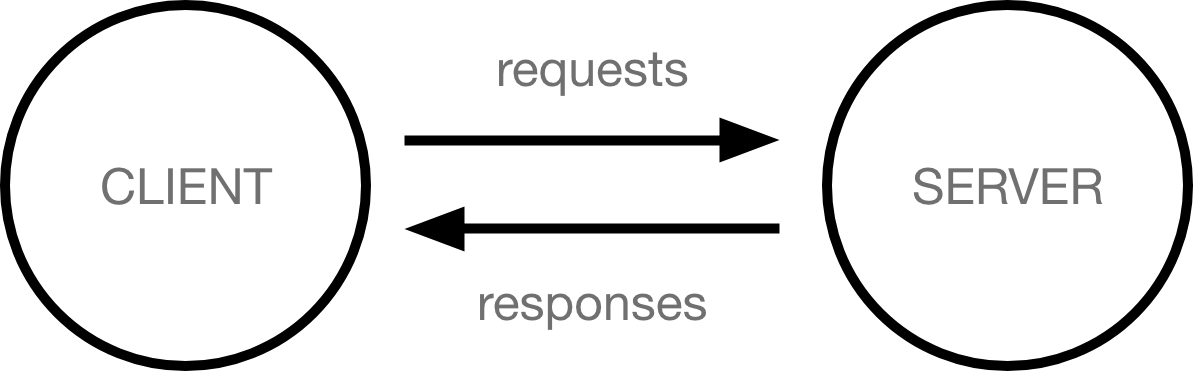
Working of Browser

Computers connected to the web are called clients and servers. A simplified diagram of how they interact might look like this:



**Client**: Clients are typically web users.

**Servers**: Servers are computers that stores web pages, sites or apps.

The web browser is no just about client and server, it also contains many others tools. Now we are to look them.

Let’s relate this working with the situation that you want to go to shop which is on the other side of the road which is in the middle of your house and the shop. While going to the shop you also have to say hello to:

**Your internet connection**: Allows you to send and receive data on the web, It's basically like the street between your house and shop.

**TCP/IP:** Transmission control protocol and Internet protocol are communication protocols that defines how data flows from the internet. This is like going to shop by bus, car or bicycle.

**DNS:** Domain naming system it is like a address book for websites. When you type a address in your URL of browser it goes to DNS for IP address. Browser wants to know the IP address so it can send the Http message to it to get data. This is like a address of shop so that you can access it.

**HTTP**: Hypertext Transfer Protocol is an application protocol that defines the language for client and server to speak with each other.

**Component files:** A website is made up of many files like HTML, CSS, and JavaScript.

**A). Working (Browser’s main functionality):**

1. The browser goes to DNS server and ask for the IP address of the server.
2. The browser sends the HTTP request message to the server, asking it to send the website copy to the user.
3. If the server accepts the request it send the response as "200 Ok" and the website copy in the form of small chunks called packets.
4. The browser assembles the packets and show the whole website to the user.

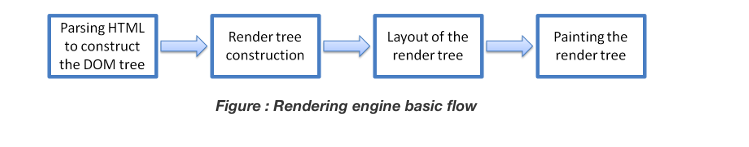
**B). the browser's main component:**

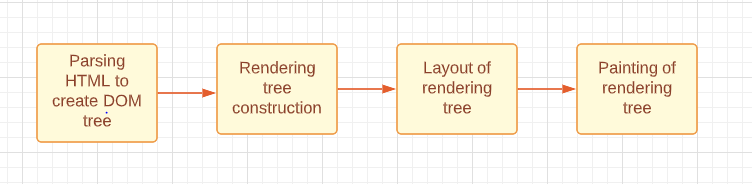
1. **User Interface:** this include all the things we can see in the browser except the window where we see the requested page.
2. **Browser Engine**: marshals actions between the UI and the rendering engine.
3. **Rendering Engine:** responsible for displaying the requested content like HTML, CSS files.
4. **Networking:** for network calls such as HTTP requests, using different implementations.
5. **UI backend:** used for drawing basic widgets, underneath it used operating system interface methods.
6. **JavaScript interpreter**: Used to parse and execute JavaScript code.
7. **Data Storage:** saving data like cookies, local Storage etc.



**C). Rendering Engine and its use:**

This component is responsible for rendering a specific web page requested by the user on their screen. It interprets HTML and XML documents along with he images that are styled by the user on their screen.



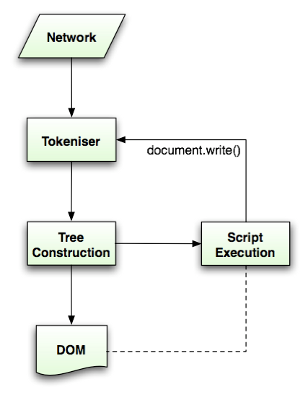


1. The HTML, CSS files are parsed and render and then converted to DOM (Document Object Model) nodes to form DOM tree.
2. Same browser also create the render tree, which orders (align) the display of elements will show inside browser.
3. Render tree goes through the layout process where the position and sizes are defined to the elements.
4. When the render tree make to its place then the paint () method calls and paint the elements by UI backend layer.

**D). Parsers:**

1. **HTML:** The work of HTML parser is to parse HTML mark-up into a parse tree.

HTML parsing flow:



The [parsing algorithm is described in detail by the HTML5 specification](http://www.whatwg.org/specs/web-apps/current-work/multipage/parsing.html). The algorithm consists of two stages:

* Tokenization
* Tree construction.

**Tokenization** is the lexical analysis, parsing the input into tokens. Among HTML tokens are start tags, end tags, attribute names and attribute values.

The tokenizer recognizes the token, gives it to the tree constructor and consumes the next character for recognizing the next token, and so on until the end of the input.

**Tree Construction:** When the parser is created the Document object is created. During the tree construction stage the DOM tree with the Document in its root will be modified and elements will be added to it. Each node emitted by the tokenizer will be processed by the tree constructor. For each token the specification defines which DOM element is relevant to it and will be created for this token. The element is added to the DOM tree, and also the stack of open elements. This stack is used to correct nesting mismatches and unclosed tags. The algorithm is also described as a state machine. The states are called "insertion modes".

**CSS:**

Well unlike HTML, CSS is a context free grammar and can be parsed using the types of parsers described in the introduction. In fact the CSS specification defines CSS lexical and syntax grammar.

Let's see some examples:   
The lexical grammar (vocabulary) is defined by regular expressions for each token:

comment \/\\*[^\*]\*\\*+([^/\*][^\*]\*\\*+)\*\/

num [0-9]+|[0-9]\*"."[0-9]+

nonascii [\200-\377]

nmstart [\_a-z]|{nonascii}|{escape}

nmchar [\_a-z0-9-]|{nonascii}|{escape}

name {nmchar}+

ident {nmstart}{nmchar}\*

"ident" is short for identifier, like a class name. "name" is an element id (that is referred by "#" )

**E). Scripts:**

The model of the web is synchronous. Authors except scripts to be parsed and executed immediately when the parser reaches a <script> tag. The parsing of the documents halts until the script has been executed. If the script is external then the resource must first be fetched from the network-this is also done synchronously, and parsing until the resource is fetched.

**F). 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.

Each renderer represents a rectangular area usually corresponding to a node's CSS box, as described by the CSS2 spec. It includes geometric information like width, height and position.

**G). Order of script processing:**

Style sheets on the other hand have a different model. Conceptually it seems that since style sheets don't change the DOM tree, there is no reason to wait for them and stop the document parsing. There is an issue, though, of scripts asking for style information during the document parsing stage. If the style is not loaded and parsed yet, the script will get wrong answers and apparently this caused lots of problems. It seems to be an edge case but is quite common. Firefox blocks all scripts when there is a style sheet that is still being loaded and parsed. Web Kit blocks scripts only when they try to access certain style properties that may be affected by unloaded style sheets.

**H). Layout:**

When the renderer is created and added to the tree, it does not have a position and size. Calculating these values is called layout or reflow.

HTML uses a flow based layout model, meaning that most of the time it is possible to compute the geometry in a single pass. Elements later ``in the flow'' typically do not affect the geometry of elements that are earlier ``in the flow'', so layout can proceed left-to-right, top-to-bottom through the document.

Layout is a recursive process. It begins at the root renderer, which corresponds to the <html> element of the HTML document. Layout continues recursively through some or all of the frame hierarchy, computing geometric information for each renderer that requires it.

**Painting:**

In the painting stage, the render tree is traversed and the renderer's "paint()" method is called to display content on the screen. Painting uses the UI infrastructure component.

Like layout, painting can also be global–the entire tree is painted–or incremental. In incremental painting, some of the renderers change in a way that does not affect the entire tree. The changed renderer invalidates its rectangle on the screen. This causes the OS to see it as a "dirty region" and generate a "paint" event. The OS does it cleverly and coalesces several regions into one. In Chrome it is more complicated because the renderer is in a different process then the main process. Chrome simulates the OS behaviour to some extent. The presentation listens to these events and delegates the message to the render root. The tree is traversed until the relevant renderer is reached. It will repaint itself (and usually its children).

**The painting order:**

CSS2 defines the order of the painting process. This is actually the order in which the elements are stacked in the stacking contexts. This order affects painting since the stacks are painted from back to front. The stacking order of a block renderer is:

1. background color
2. background image
3. border
4. children
5. outline