Waiting for new pages to load takes time. People want web sites to be faster and more responsive. The technology that makes this possible is a programming technology called Ajax.

Ajax lets a web page ask for and receive a response from a web server and then update itself without having to load a new web page. The result is that the web page feels more responsive.

Example: <http://maps.google.com> You can zoom into the map, move north and even grab the map and drag it around.

What is Ajax

Ajax stands for *Asynchronous* *Javascript* and *XML* but it isn’t an official technology like HTML. It’s a term that refers to the interaction of a mix of technologies – Javascript, the web browser and the web server – to retrieve and display new content without loading a new web page.

Things you can do with Ajax:

Display new HTML content *without reloading the page*

Submit a form and instantly display results

Log in without leaving the page

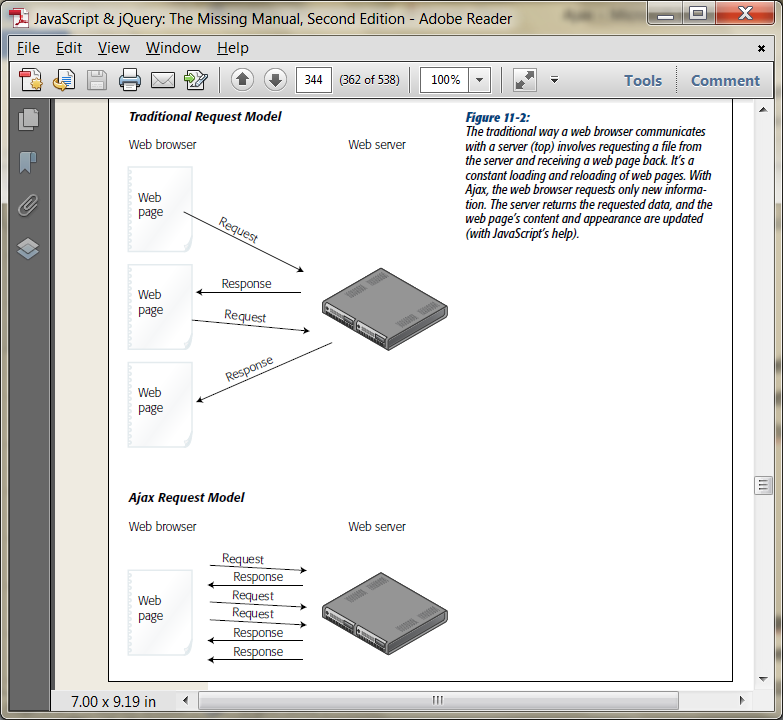
Star-rating widgets

Browsing through database information (e.g. Twitter tweets – when you scroll down more tweets are loaded)

It’s the “without reloading the webpage” that’s the cool bit that makes the page more responsive and improves the user’s experience – in fact Ajax lets you create websites that feel more like desktop programs.

Ajax – The basics

The traditional model of web browser communication with a server involves requesting a file from the server and receiving a web page back. It’s a constant loading and reloading of web pages. With Ajax, the web browser requests only new information. The server returns the requested data, and the web page’s content and appearance are updated (with javascript’s help)



Pieces of the Puzzle

Ajax brings together three different components:

**The web browser**. It has a special built in ***XMLHttpRequest*** object which makes Ajax work. This object is what lets Javascript talk to the web server and receive information in response. It was initially introduced into IE5 and has since made its way into all major browsers.

**Javascript**. This does all the heavy lifting in Ajax. It sends a request to the web server, waits for a response, processes the response, and usually updates the web page (using the DOM to add, change and remove HTML tags and content)

**The Web Server**. This receives the requests from and sends information back to the web browser. The server might return some HTML or plain text, or it might return an XML document or JSON data.

The Web Server usually involves several different types of technologies including a web server (e.g. Apache), application server that understands a server-side language like PHP, Java, C#, Ruby etc (e.g. Tomcat) and database server (e.g. Oracle)

There are many different combinations of web servers, application servers and database servers. E.g. Microsoft IIS with ASP.net and SQL server or Apache, PHP and MySQL (AMP)

MEAN (Mongo, Express, Angular, Node)

Ajax works with a web server. The examples later will work locally but we will also host the files on a web server using XAMPP.

Talking to the Web Server

The core of any Ajax program is the *XMLHTTPRequest* object. Sometimes just referred to as XHR, the XMLHttpRequest object is a feature built into current web browsers that allows JavaScript to send information to a web server and receive information in return. There are five steps, all of which can be accomplished in Javascript:

1. Create an instance of the *XMLHttpRequest* object. In its most basic form it looks like this

var newXHR = new XMLHttpRequest();

**Unfortunately, there are enough cross-browser problems with Ajax that it is best to use a Javascript library – like jQuery – to make your Ajax request**

1. Use the XHR’s open() method to specify what kind of data you’ll send and where the data will go

You can send the data in two ways, using either GET or POST methods – the same options you use with HTML forms - the GET method sends any information to the web server as part of the URL – *shop.php?productID=34* for example. The data appears as name/value pairs after the ?. Think of the name as the name of a field on a form and the value as what the user types in.

**NOTE: for security reasons browsers won’t let you make Ajax requests to other domains – the URL you specify in the open()method must be on the same website as the page making the request**

The POST method sends data separately from the URL. Usually you use the GET method to get data back from the server, and the POST method to update information on the server. You also use the open() method to specify the page on the server the data is sent to. (That’s usually a page on the web server that uses a server side scripting language like PHP to retrieve data from a database etc and you point to it by its URL.

For example, the following code tells the XHR object what method to use (GET) and which page on the server to request

**newXHR.open(‘POST’, ‘search.php?productID=34’);**

1. **Create a function to handle the results.** When the web server returns a result like new database information etc. you will want to do something with the result. You need to write a Javascript function to deal with the result – this function (called a *callback* function) is often the meat of your program.

Usually this function will manipulate the page’s content by removing elements, adding elements, or changing elements. (JQuery will handle the details)

1. **Send the Request.** To actually send information to the web server, you use the XHR object’s send() method. Everything upto this point is just setup – this step is what tells the web browser to send the request. If you are using the GET method, this step is as simple as

**newXHR.**send(null);

The null part indicates that you are not sending any additional data.

Remember, with the GET method, the data is sent in the URL like this:

search.php?q=javascript (where q=javascript is the data)

With the POST method, you must provide the data along with the send() method like this:

newXHR.send(‘q=javascript’);

Once the request is sent, your javascript program doesn’t necessarily stop. The ‘A’ in Ajax stands for asynchronous, which means the Javascript program can continue doing other things – the web browser doesn’t just sit around and wait for the server to respond

1. Receive the Response

After the server has processed the request, it sends back a response to the web browser. Actually, the callback function you created in step 3 handles the response, but meanwhile, the XHR object receives several pieces of information from the Web Server, including the status of the request, a text response, and possibly an XML response.

The status is a number: 404 = file not found, 200=ok, 500=internal server error, 403=access forbidden

In addition you usually receive a text response stored in the XHR object’s *responseText* property. This response can be a chunk of HTML, a simple text message, or a complex set of JSON data.

Finally if the server responds with an XML file, it’s stored in the XHRs *responseXML* property.

Whatever data the server returns, its available to the callback function to use to update the web page. Once the callback function finishes up, the entire Ajax cycle is over

Example: <http://www.w3schools.com/php/php_ajax_livesearch.asp>

Examples: http://www.datatables.net/examples/data\_sources/ajax.html.