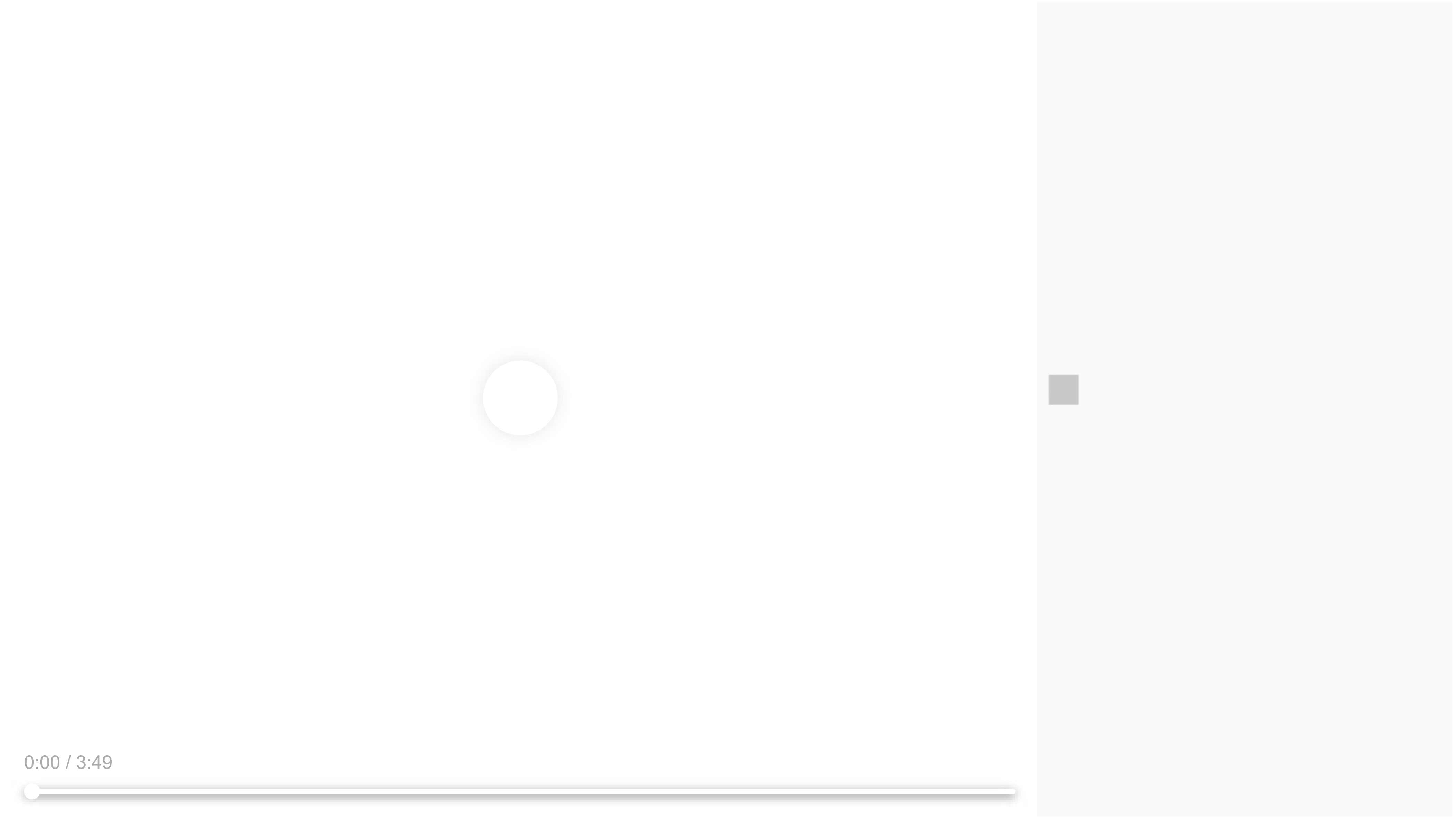
AJAX | Day 1 | Pre-Work

AJAX

Table of contents:

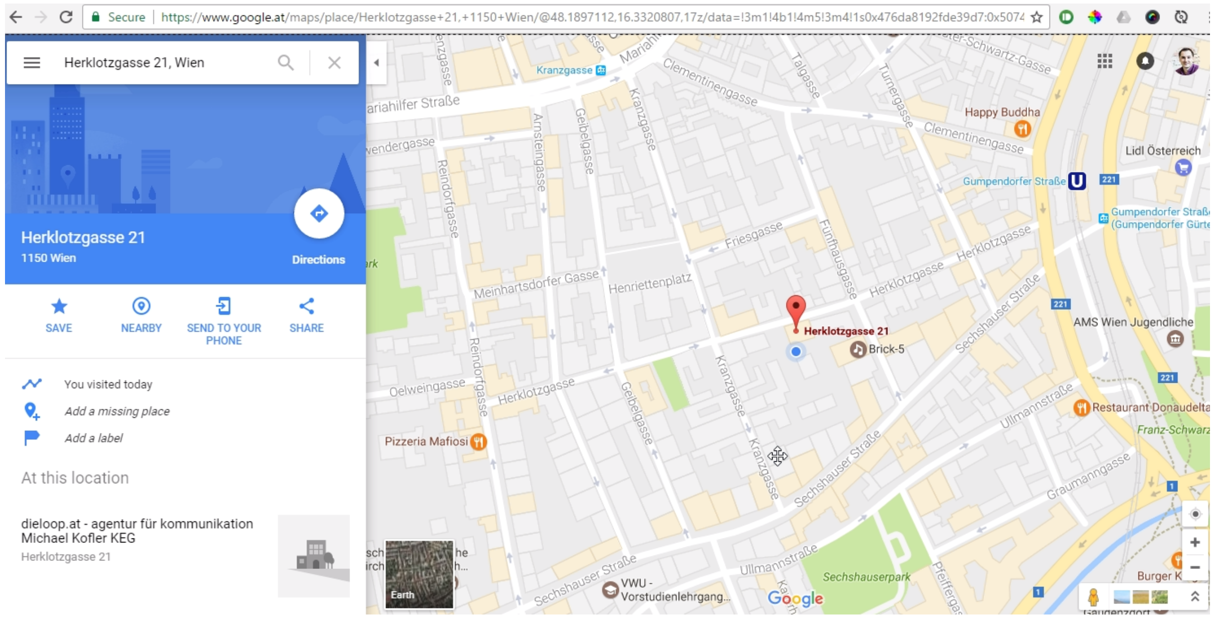
JSON

Ajax the jQuery way



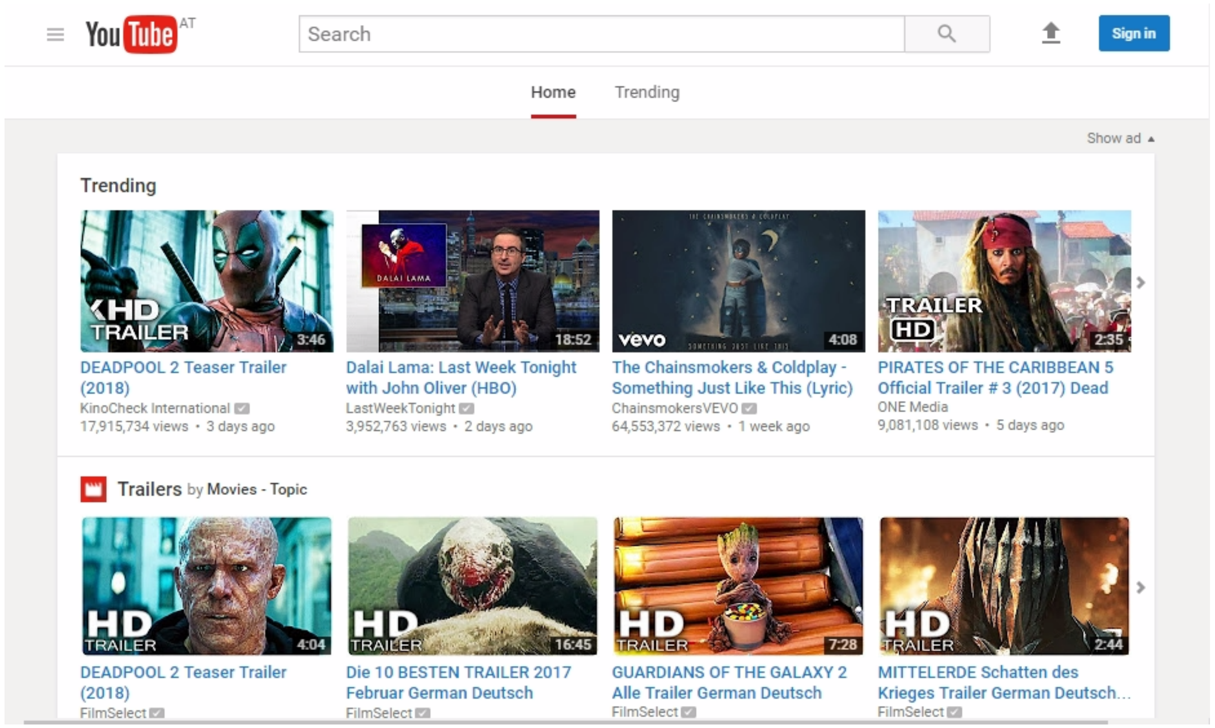
What is AJAX?

AJAX stands for Asynchronous JavaScript and XML. In simple terms, it means using a set of methods built in to JavaScript to transfer data between the browser and a server in the background. The term Ajax was first coined in 2005. An excellent example of this technology is Google Maps, in which new sections of a map are downloaded from the server when needed, without requiring a page refresh.
Using Ajax not only substantially reduces the amount of data that must be sent back and forth, but also makes web pages seamlessly dynamic—allowing them to behave more like self-contained applications. The results are a much improved user interface and better responsiveness.
AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages. Until now you should know HTML, CSS, JavaScript, and today you will also learn XML, then you will have everything for creating interactive web applications.
Since Ajax can be a useful approach for making web applications with a rich behavior, it is commonly used in some Web 2.0 applications. It is important to understand that Ajax and Web 2.0 are not the same thing. Ajax doesn't have anything to do with the read/write web, which is what web 2.0 is about. But it is true that these two technologies are complementing each other very well.



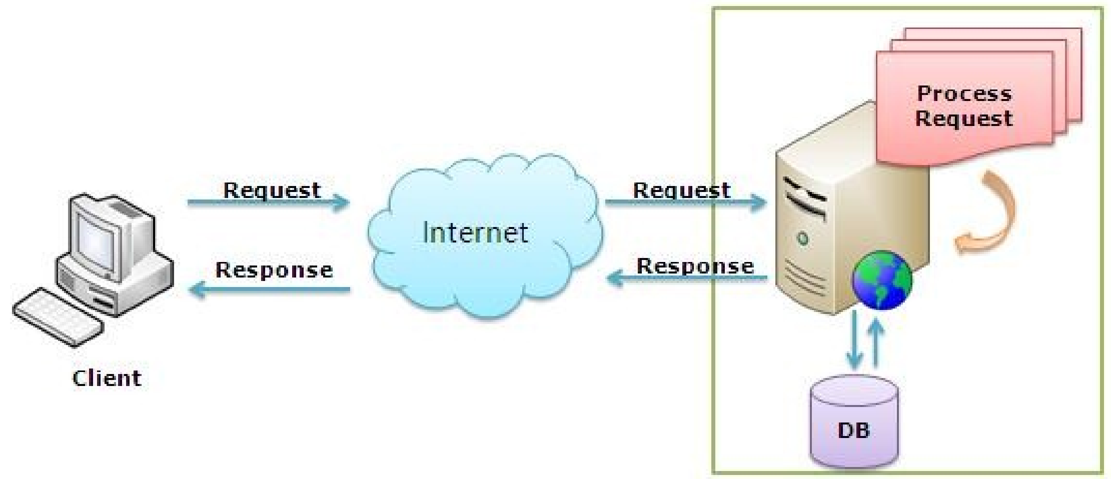
Web 2.0

Web 2.0 refers to a perceived second-generation of Web based communities and hosted services, such as social networking sites, wikis and folksonomies, that facilitate collaboration and sharing between users. A Web 2.0 website may exhibit some basic common characteristics. The most important one is so called “Internet as platform” and it means delivering and allowing users to use applications entirely through a browser. Also users are owning the data on the site and exercising control over it. Another common thing in Web 2.0 websites is the architecture of participation and democracy that encourages users to add value to the application as they use it. This stands in sharp contrast to hierarchical access-control in applications, in which systems categorize users into roles with varying levels of functionality. Democracy is one of the main things in the new generation applications. It is about giving users the freedom to change or add information as they are using the application. In the recent years it was proven to be a successful decision because users like to have a little bit of control over things. The collaboration of Ajax and Web 2.0 is a perfect example of the development and progress of web technologies in completely different and innovative direction. Web applications are starting to act more like standalone applications and websites are becoming more dynamic, interactive and user-friendly.


YouTube AT
Search
Sign in
Home
Trending
Show ad
Trending
DEADPOOL 2 Teaser Trailer (2018)
KinoCheck International
17,915,734 views • 3 days ago
Dalai Lama: Last Week Tonight with John Oliver (HBO)
LastWeekTonight
3,952,763 views • 2 days ago
The Chainsmokers & Coldplay - Something Just Like This (Lyric)
ChainsmokersVEVO
64,553,372 views • 1 week ago
PIRATES OF THE CARIBBEAN 5 Official Trailer # 3 (2017) Dead
ONE Media
9,081,108 views • 5 days ago
Trailers by Movies - Topic
DEADPOOL 2 Teaser Trailer (2018)
FilmSelect
Die 10 BESTEN TRAILER 2017 Februar German Deutsch
FilmSelect
GUARDIANS OF THE GALAXY 2 Alle Trailer German Deutsch
FilmSelect
MITTELERDE Schatten des Krieges Trailer German Deutsch...
FilmSelect

A little history of AJAX

The beginnings of Ajax as used today started with the release of Internet Explorer 5 in 1999, which introduced a new ActiveX object, XMLHttpRequest. ActiveX is Microsoft's technology for signing plugins that install additional software to your computer. Other browser developers later followed suit, but rather than using ActiveX, they all implemented the feature as a native part of the JavaScript interpreter. However, even before then, an early form of Ajax had already surfaced that used hidden frames on a page that interacted with the server in the background. Chat rooms were early adopters of this technology, using it to poll for and display new message posts without requiring page reloads. Ajax became popular in 2005 by Google search engine (with Google Suggest). With Ajax you can create better, faster, and more user-friendly web applications. This technique is based on JavaScript and HTTP requests. With Ajax, your JavaScript can communicate directly with the server, using the JavaScript XMLHttpRequest object. With this object, your JavaScript can trade data with a web server, without reloading the page. Ajax uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages as you see on the image below. This is intended to increase the web page's interactivity, speed and usability. So let’s see how to implement Ajax by using JavaScript.



In next seven steps you can see the explanation from image above:

- 1. An event occurs in a web page (the page is loaded, a button is clicked)
- 2. An XMLHttpRequest object is created by JavaScript
- 3. The XMLHttpRequest object sends a request to a web server
- 4. The server processes the request (communicate with the Database if it necessary)

- 5. The server sends a response back to the web page
- 6. The response is read by JavaScript
- 7. Proper action (like page update) is performed by JavaScript

XMLHttpRequest Object

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 support the XMLHttpRequest object. Because of the differences between browser implementations of XMLHttpRequest, you can create a special function in order to ensure that your code will work on all major browsers (old Internet explorer versions - IE5 and IE6).
To do this, you must understand the three ways of creating an XMLHttpRequest object:

- IE 5: request = new ActiveXObject("Microsoft.XMLHTTP")
- IE 6+: request = new ActiveXObject("Msxml2.XMLHTTP")
- All others: request = new XMLHttpRequest()

This is the case because Microsoft chose to implement a change with the release of Internet Explorer 6, while all other browsers use a slightly different method. Therefore, the code below will work for all major browsers released over the last years.

```
<script>
function ajaxRequest()
{
try // Non IE Browser?
{ // Yes
var request = new XMLHttpRequest()
}
catch(e1)
{
try // IE 6+?
{ // Yes
request = new ActiveXObject("Msxml2.XMLHTTP")
}
catch(e2)
{
try // IE 5?
{ // Yes
request = new ActiveXObject("Microsoft.XMLHTTP")
}
catch(e3) // There is no AJAX Support
{
request = false
}
}
}
return request
}
</script>
```

You may remember the introduction to error handling in JavaScript, using the try...catch construct. Example above is a perfect illustration of its utility, because it uses the try keyword to execute the non-IE Ajax command, and upon success, jumps on to the final return statement, where the new object is returned.
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.

To run the function from the example below you will need a web server. For this example you will also need to create your_doc.txt file and insert some content inside it. When you press the button “Change content”, content from your_doc.txt will be loaded in the HTML page over the XMLHttpRequest object.

```
<!DOCTYPE html>
<html>
<body>
<div id="content">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change
Content</button>
</div>
<script>
function loadDoc() {
 var xhttp = new XMLHttpRequest();
 xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
     document.getElementById("content").innerHTML =
     this.responseText;
   }
 };
 xhttp.open("GET", "your_doc.txt", true);
 xhttp.send();
}
</script>
</body>
</html>
```

new XMLHttpRequest() - Creates a new XMLHttpRequest object

Onreadystatechange - Defines a function to be called when the readyState property change

readyState - Holds the status of the XMLHttpRequest:

0: request not initialized

1: server connection established

2: request received

3: processing request

4: request finished and response is ready

status - Returns the status-number of a request:

200: "OK"

403: "Forbidden"

404: "Not Found"

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

open(method, url, async) - Specifies the type of request

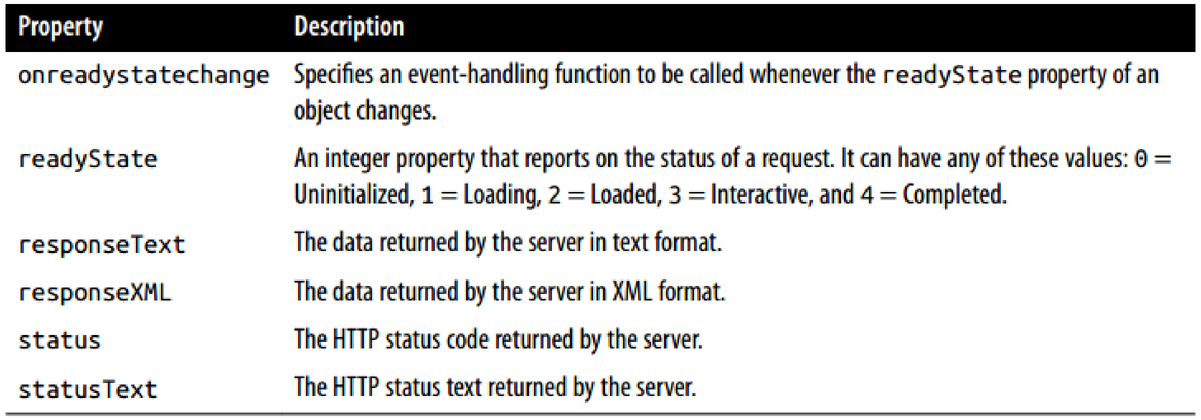
method: the type of request: GET or POST

url: the server (file) location

async: true (asynchronous) or false (synchronous)

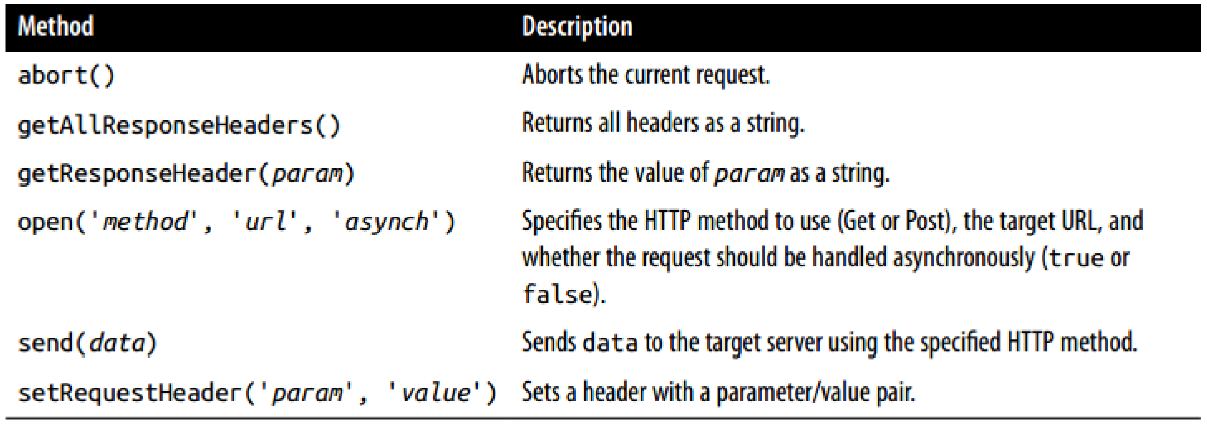
send() - Sends the request to the server (used for GET)

XMLHttpRequest Object Methods



Property	Description
`onreadystatechange`	Specifies an event-handling function to be called whenever the `readyState` property of an object changes.
`readyState`	An integer property that reports on the status of a request. It can have any of these values: `0` = Uninitialized, `1` = Loading, `2` = Loaded, `3` = Interactive, and `4` = Completed.
`responseText`	The data returned by the server in text format.
`responseXML`	The data returned by the server in XML format.
`status`	The HTTP status code returned by the server.
`statusText`	The HTTP status text returned by the server.

An XMLHttpRequest object’s properties



Method	Description
`abort()`	Aborts the current request.
`getAllResponseHeaders()`	Returns all headers as a string.
`getResponseHeader(`*`param`*`)`	Returns the value of *`param`* as a string.
`open('`*`method`*`', '`*`url`*`', '`*`asynch`*`')`	Specifies the HTTP method to use (Get or Post), the target URL, and whether the request should be handled asynchronously (`true` or `false`).
`send(`*`data`*`)`	Sends `data` to the target server using the specified HTTP method.
`setRequestHeader('`*`param`*`', '`*`value`*`')`	Sets a header with a parameter/value pair.

An XMLHttpRequest object's methods

Data formats

The response to an Ajax request usually comes in one of three formats: HTML, XML, or JSON. Below is a comparison of these formats.

HTML

You are most familiar with HTML, and, when you want to update a section of a web page, it is the simplest way to get data into a page - from that reason here we would not explain HTML again, we can just say benefits of this format:

- It is easy to write, request and display
- The data sent from the server goes straight to the page. There's no need for the browser to process data (as with the other two formats)

XML

XML stands for **eXtensible Markup Language**. XML was designed to store and transport data, it was designed to be both human- and machine-readable. As a developer you need to understand how computers store and access data. For our purposes, computers understand two kinds of data files: **binary files and text files**.

A binary file, at its simplest, is just a stream of bits (1 and 0). It’s up to the application that created a binary file to understand what all of the bits mean. That’s why binary files can only be read and produced by certain computer programs, which have been specifically written to understand them.
Like binary files, text files are also streams of bits. However, in a text file these bits are grouped together in standardized ways, so that they always form numbers. These numbers are then further mapped to characters that are human readable. Because of these standards, text files can be read by many applications, and can even be read by humans, using a simple text editor (application).
A very well-known language based on the SGML work is the HyperText Markup Language (HTML). HTML uses many of SGML’s concepts to provide a universal markup language for the display of information, and the linking of different pieces of information. The idea was that any HTML document (or web page) would be presentable in any application that was capable of understanding HTML (termed a web browser).
XML is a subset of SGML as HTML, with the same goals (markup of any type of data), but with as much of the complexity eliminated as possible. It is important (for all types of software developers!) to have a good understanding of XML. Now we can say that XML is a software- and hardware-independent Markup Language for storing and transporting data.
Now let’s see a real XML example and how simple it actually it. This example is an email to Lio from Nicky, stored as XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<email>
 <to>Lio</to>
 <from>Nicky</from>
 <heading>Reminder</heading>
 <body>We have classwork today!</body>
</email>
```

The first line defines XML document, you know <!DOCTYPE html> from HTML documents? This is the same thing just for XML.
XML was designed to carry data - with the focus on what data is. **HTML was designed to display data** - with focus on how data looks. XML tags are not predefined like HTML tags are, so now you know that XML language has no predefined tags and in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by us. In XML, the author must define both the tags and the document structure.
XML separates data from presentation and does not carry any information on how it will be displayed. The same XML data can be used in many different presentation scenarios. Most XML applications will work as expected even if new data is added (or removed)

which means that XML is extensible.

Here we have have an XML example for a book catalog:

```
<?xml version="1.0"?>
<catalog>
 <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2016-10-01</publish_date>
    <description>An in-depth look at creating
applications
    with XML.</description>
 </book>
 <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2015-12-16</publish_date>
    <description>A former architect battles corporate
zombies,
    an evil sorceress, and her own childhood to become
queen
    of the world.</description>
 </book>
 <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology
    society in England, the young survivors lay the
    foundation for a new society.</description>
 </book>
```

```xml
 <book id="bk104">
    <author>Corets, Eva</author>
    <title>Oberon's Legacy</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
    <description>In post-apocalypse England, the
mysterious
    agent known only as Oberon helps to create a new life
    for the inhabitants of London. Sequel to Maeve
    Ascendant.</description>
 </book>
 <book id="bk105">
    <author>Corets, Eva</author>
    <title>The Sundered Grail</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-09-10</publish_date>
    <description>The two daughters of Maeve, half-
sisters,
    battle one another for control of England. Sequel to
    Oberon's Legacy.</description>
 </book>
 <book id="bk106">
    <author>Randall, Cynthia</author>
    <title>Lover Birds</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-09-02</publish_date>
    <description>When Carla meets Paul at an ornithology
    conference, tempers fly as feathers get ruffled.
</description>
 </book>
 <book id="bk107">
    <author>Thurman, Paula</author>
    <title>Splish Splash</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>A deep sea diver finds true love twenty
    thousand leagues beneath the sea.</description>
 </book>
 <book id="bk108">
    <author>Knorr, Stefan</author>
    <title>Creepy Crawlies</title>
    <genre>Horror</genre>
    <price>4.95</price>
    <publish_date>2000-12-06</publish_date>
    <description>An anthology of horror stories about
roaches,
    centipedes, scorpions  and other insects.
</description>
 </book>
 <book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertant trip through a
Heisenberg
    Uncertainty Device, James Salway discovers the
problems
    of being quantum.</description>
 </book>
 <book id="bk110">
    <author>O'Brien, Tim</author>
    <title>Microsoft .NET: The Programming Bible</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-09</publish_date>
    <description>Microsoft's .NET initiative is explored
in
    detail in this deep programmer's reference.
```

```
</description>
 </book>
 <book id="bk111">
    <author>O'Brien, Tim</author>
    <title>MSXML3: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-01</publish_date>
    <description>The Microsoft MSXML3 parser is covered
in
    detail, with attention to XML DOM interfaces, XSLT
processing,
    SAX and more.</description>
 </book>
 <book id="bk112">
    <author>Galos, Mike</author>
    <title>Visual Studio 7: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>49.95</price>
    <publish_date>2001-04-16</publish_date>
    <description>Microsoft Visual Studio 7 is explored in
depth,
    looking at how Visual Basic, Visual C++, C#, and ASP+
are
    integrated into a comprehensive development
    environment.</description>
 </book>
</catalog>
```

Now we will load our book collection into an HTML page using AJAX, here you can see the example:

```
<!DOCTYPE html>
<html>
<style>
table,
th,
td {
   border: 1px solid black;
   border-collapse: collapse;
}
th,
td {
   padding: 5px;
}
</style>

<body>
   <h1>The XMLHttpRequest Object</h1>
   <button type="button" onclick="loadDoc()">Get my books
collection</button>
   <br>
   <br>
   <table id="content"></table>
   <script>
   function loadDoc() {
       var xhttp = new XMLHttpRequest();
       xhttp.onreadystatechange = function() {
           if (this.readyState == 4 && this.status ==
200) {
               myFunction(this);
           }
       };
       xhttp.open("GET", "books.xml", true);
       xhttp.send();
   }
   function myFunction(xml) {
       var i;
       var xmlDoc = xml.responseXML;
       var table = "<tr><th>Author</th><th>Title</th>
</tr>";
       var x = xmlDoc.getElementsByTagName("book");
       for (i = 0; i < x.length; i++) {
           table += "<tr><td>" +
                x[i].getElementsByTagName("author")
[0].childNodes[0].nodeValue +
                "</td><td>" +
                x[i].getElementsByTagName("title")
[0].childNodes[0].nodeValue +
                "</td></tr>";
       }
       document.getElementById("content").innerHTML =
table;
   }
   </script>
</body>
</html>
```

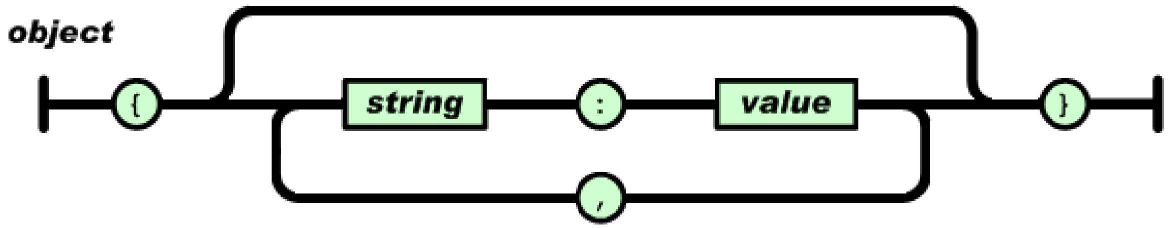
JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python and many others. These properties make JSON an ideal data-interchange language.
These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.
When exchanging data between a browser and a server, the data can only be text. JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
We can also convert any JSON received from the server into JavaScript objects.
This way we can work with the data as JavaScript objects, with no complicated parsing and translations.
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

In JSON, they take on these forms:
An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

```
var myObj = { "name":"Nicolaus", "age":31,
"city":"Vienna" };
var myJSON = JSON.stringify(myObj);
```

If you receive data in JSON format, you can convert it into a JavaScript object:

```
var myObj = { "name":"Nicolaus", "age":31,
"city":"Vienna" };
document.getElementById("content").innerHTML =
myObj.name;
var myObj = JSON.parse(myJSON);
```

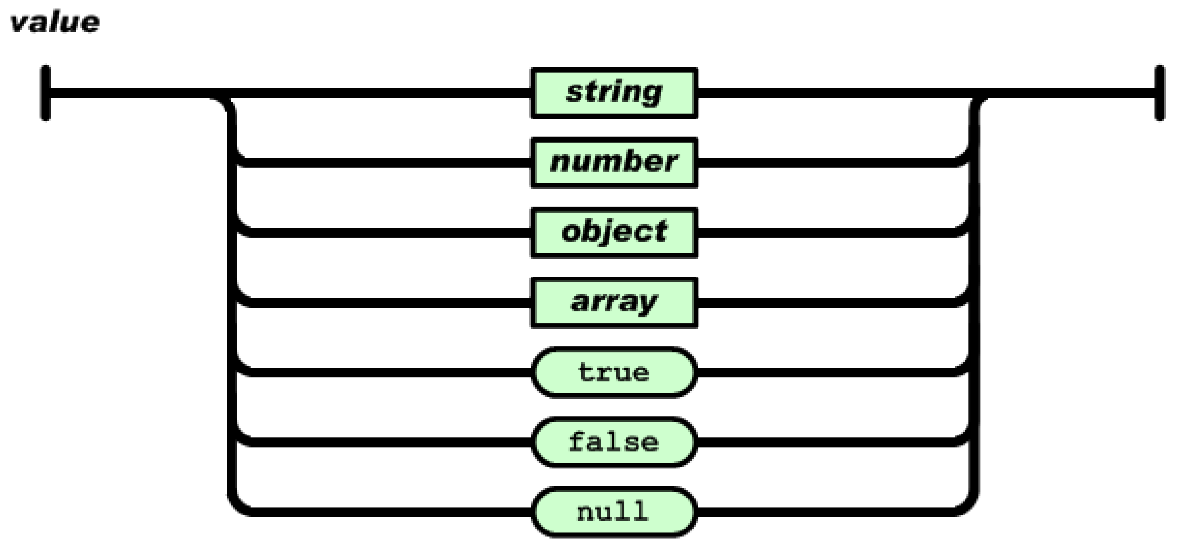
JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:
JSON.parse()
So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.
When storing data, the data has to be a certain format and regardless of where you choose to store it, text is always one of the legal formats.

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- Null

JSON values **CANNOT** be one of the following data types:

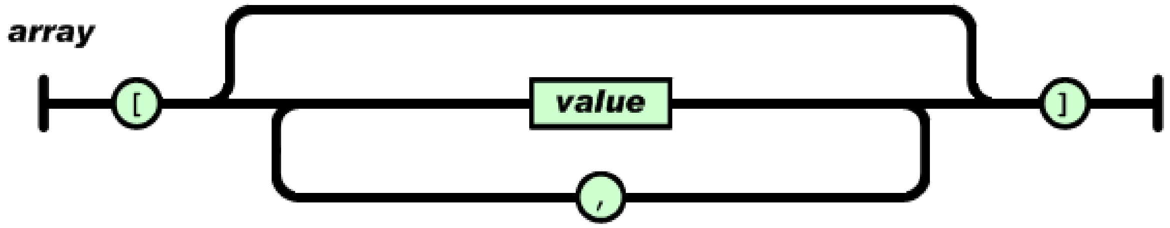
- a function
- a date
- undefined



JSON makes it possible to store JavaScript objects as text. In the next example you can see how to store data in local storage and how to retrieve it:

```
<!DOCTYPE html>
<html>
<body>
   <h2>Store and retrieve data from local storage.</h2>
   <p id="demo"></p>
   <script>
   var myObj, myJSON, text, obj;
   //Storing data:
   myObj = {
       "name": "Nikolaus",
       "age": 31,
       "city": "Vienna"
   };
   myJSON = JSON.stringify(myObj);
   localStorage.setItem("testJSON", myJSON);
   //Retrieving data:
   text = localStorage.getItem("testJSON");
   obj = JSON.parse(text);
   document.getElementById("demo").innerHTML = obj.age;
   </script>
</body>
</html>
```

An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).



A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested. Arrays in JSON are almost the same as arrays in JavaScript. In JSON, array values must be of type string, number, object,

array, boolean or null.
In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and undefined.

```
students = [ "Nathy", "Tamara", "Lio", "Nicky", "Moumen",
"Nikolaus" ];
```

```
<!DOCTYPE html>
<html><body>
   <p>Access an array value of a JSON object.</p>
   <p id="student"></p>
   <script>
   students= ["Nathy", "Tamara", "Lio", "Nicky", "Moumen",
"Nikolaus"];
   document.getElementById("student").innerHTML =
students[5];
   </script>
</body>
</html>
```

Ajax the jQuery way

Using jQuery helps you to write your code faster and avoid a range of problems with the browser compatibility. jQuery also creates and sends the request. jQuery has several methods for working with Ajax.
You can find all of them here: http://api.jquery.com/category/ajax/

This is a good example for inserting data to the database using an Ajax request

First we have to create a database:

```
CREATE DATABASE ajaxtest ;
```

After that we will create a simple table and name it “example” :

```
CREATE TABLE `example` (
ID INT(5) NOT NULL AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(20) NOT NULL
)
```

Now open the htdocs folder (C:/xampp/htdocs/) and create a new folder and give it a name (Testing). Then we will make an index.html file inside the folder :

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>


</head>
<body>

<form id="foo">
   <label for="bar">A bar</label>
   <input id="bar" name="bar" type="text" value="" />

   <input type="submit" value="Send" />
</form>


<script>
// Variable to hold request
```

```
var request;

// Bind to the submit event of our form
$("#foo").submit(function(event){

   // Prevent default posting of form - put here to work in case of errors
   event.preventDefault();

   // Abort any pending request
   if (request) {
       request.abort();
   }
   // setup some local variables
   var $form = $(this);

   // Let's select and cache all the fields
   var $inputs = $form.find("input, select, button, textarea");

   // Serialize the data in the form
   var serializedData = $form.serialize();

   // Let's disable the inputs for the duration of the Ajax request.
   // Note: we disable elements AFTER the form data has been serialized.
   // Disabled form elements will not be serialized.
   $inputs.prop("disabled", true);

   // Fire off the request to /form.php
   request = $.ajax({
       url: "/Testing/form.php",
       type: "post",
       data: serializedData
   });

   // Callback handler that will be called on success
   request.done(function (response, textStatus, jqXHR){
       // Log a message to the console
       console.log("Hooray, it worked!");
```

```
    });

    // Callback handler that will be called on failure
    request.fail(function (jqXHR, textStatus, errorThrown){
        // Log the error to the console
        console.error(
            "The following error occurred: "+
            textStatus, errorThrown
        );
    });

    // Callback handler that will be called regardless
    // if the request failed or succeeded
    request.always(function () {
        // Reenable the inputs
        $inputs.prop("disabled", false);
    });
});
</script>
</body>
</html>
```

Now we will make a form.php file

```php
<?php
// You can access the values posted by jQuery.ajax
// through the global variable $_POST, like this:
$bar = isset($_POST['bar']) ? $_POST['bar'] : null;

$host= "localhost";
$username="root";
$password="";
$dbname="ajaxtest";

$conn =
mysqli_connect($host,$username,$password,$dbname);

if($conn){
        echo "success";
}

$query= "INSERT INTO `example` (name) VALUES ('$bar');";
if(mysqli_query($conn,$query)){
        echo "insert success";
}
?>
```

Last modified: Tuesday, 19 June 2018, 11:43 AM