<https://www.w3schools.com/js/js_ajax_http_send.asp>

Let’s break this down into its two parts:

* **AJAX** — *AJAX* stands for Asynchronous Javascript and XML, it’s used to allow web pages (*client-side*) to update *asynchronously* by communicating with a **web server** and by exchanging data. This essentially means that applications can talk to a server in the background of the application. It uses some core components to function:
  1. The browser’s XMLHttpRequest object to request data from a server
  2. HTML/CSS to display or collect data
  3. DOM for dynamic display
  4. JSON/XML for interchanging the data
  5. Javascript to unify everything
* Notice that AJAX is therefore not a singular tool but a series of tools that when used in combination can allow a web application to communicate with a server and show some data.\
* **Node.js**— Node is a server-side application that allows you to write your server in Javascript (plus some other nifty feature like nonblocking I/O).

**Therefore**, a website will use AJAX to communicate with a server, but that server may not necessarily be running Node. The server could be running on tons of potential frameworks and languages: PHP, Ruby on Rails, Python, but that doesn’t matter as AJAX is specific to the *client’s* request and not the framework of the server, but that framework might just happen to be *Node*.

What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works



* 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
* 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

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a web 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 (Chrome, Firefox, IE7+, Edge, Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

var xhttp = new XMLHttpRequest();

## Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means that both the web page and the XML file it tries to load, must be located on the same server.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server.

**Older Browsers (IE5 and IE6)**

Old versions of IE (5/6) use an ActiveX object instead of the XMLHttpRequest object:

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

To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveX object

if (window.XMLHttpRequest) {  
   // code for modern browsers  
   xmlhttp = new XMLHttpRequest();  
 } else {  
   // code for old IE browsers  
   xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
}

## XMLHttpRequest Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| new XMLHttpRequest() | Creates a new XMLHttpRequest object |
| abort() | Cancels the current request |
| getAllResponseHeaders() | Returns header information |
| getResponseHeader() | Returns specific header information |
| open(method, url, async, user, psw) | Specifies the request  method: the request type GET or POST url: the file location async: true (asynchronous) or false (synchronous) user: optional user name psw: optional password |
| send() | Sends the request to the server Used for GET requests |
| send(string) | Sends the request to the server. Used for POST requests |
| setRequestHeader() | Adds a label/value pair to the header to be sent |

## XMLHttpRequest Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| 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 |
| responseText | Returns the response data as a string |
| responseXML | Returns the response data as XML data |
| status | Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp) |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

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:

xhttp.open("GET", "ajax\_info.txt", true);  
xhttp.send();

|  |  |
| --- | --- |
| **Method** | **Description** |
| 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) |
| send(*string*) | Sends the request to the server (used for POST) |

## 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

**Example**

xhttp.open("GET", "demo\_get.asp", true);  
xhttp.send();

<!DOCTYPE html>

<html>

<body>

<h2>The XMLHttpRequest Object</h2>

<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("POST", "demo\_post.asp", true);

xhttp.send();

}

</script>

</body>

</html>

In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

xhttp.open("GET", "demo\_get.asp?t=" + Math.random(), true);  
xhttp.send();

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

xhttp.open("GET", "demo\_get2.asp?fname=Henry&lname=Ford", true);  
xhttp.send();

## POST Requests

A simple POST request:

### **Example**

xhttp.open("POST", "demo\_post.asp", true);  
xhttp.send();

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

### **Example**

xhttp.open("POST", "ajax\_test.asp", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("fname=Henry&lname=Ford");

function Load\_dbProfiles() {

var SwitchNo = $('#dbSwitch').val();

if (SwitchNo == "0") return;

$.ajax({

type: "POST",

url: "SearchRunLog.aspx/Get\_Data\_dbProfiles",

data: "{opcion: '" + SwitchNo + "'}",

contentType: "application/json; charset=utf-8",

dataType: "json",

async: true,

success: function (response) {

var dbProfile = document.getElementById("dbProfile");

dbProfile.innerHTML = "";

var option = document.createElement("option");

option.val = "0";

option.text = "Select One";

dbProfile.add(option);

for (var i = 0; i < response.d.length; i++) {

var option2 = document.createElement("option");

option2.val = response.d[i].ids;

option2.text = response.d[i].names;

dbProfile.add(option2);

}

}

});

}

|  |  |
| --- | --- |
| **Method** | **Description** |
| setRequestHeader(*header, value*) | Adds HTTP headers to the request  *header*: specifies the header name *value*: specifies the header value |

## The url - A File On a Server

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

xhttp.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?

Server requests should be sent asynchronously.

The async parameter of the open() method should be set to true:

xhttp.open("GET", "ajax\_test.asp", true);

By sending asynchronously, 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 after the response is ready

## The onreadystatechange Property

With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.

The function is defined in the onreadystatechange property of the XMLHttpResponse object:

### **Example**

xhttp.onreadystatechange = function() {  
  if (this.readyState == 4 && this.status == 200) {  
    document.getElementById("demo").innerHTML = this.responseText;  
  }  
};  
xhttp.open("GET", "ajax\_info.txt", true);  
xhttp.send();

You will learn more about onreadystatechange in a later chapter.

## Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

xhttp.open("GET", "ajax\_info.txt", false);

Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an onreadystatechange function:

### **Example**

xhttp.open("GET", "ajax\_info.txt", false);  
xhttp.send();  
document.getElementById("demo").innerHTML = xhttp.responseText;