AJAX is not a programming language.

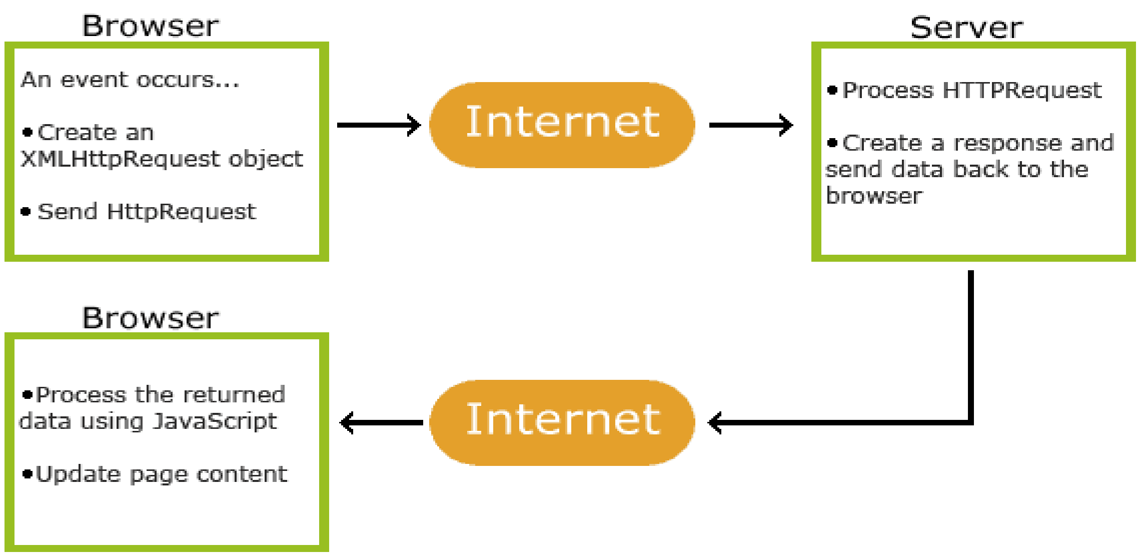
AJAX just uses a combination of:

- 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


Browser
An event occurs...
Create an XMLHttpRequest object
Send HttpRequest
Internet
Server
Process HTTPRequest
Create a response and send data back to the browser
Browser
Process the returned data using JavaScript
Update page content
Internet

- 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

Modern Browsers (Fetch API)

Modern Browsers can use Fetch API instead of the XMLHttpRequest Object.

The Fetch API interface allows web browser to make HTTP requests to web servers.

If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way.

AJAX - The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object.

1. Create an XMLHttpRequest object
2. Define a callback function
3. Open the XMLHttpRequest object
4. Send a Request to a server

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.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE, Edge, Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

Define a Callback Function

A callback function is a function passed as a parameter to another function.

In this case, the callback function should contain the code to execute when the response is ready.

```
xhttp.onload = function() {
  // What to do when the response is ready
}
```

Send a Request

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

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

Example

```
// Create an XMLHttpRequest object
const xhttp = new XMLHttpRequest();
```

```
// Define a callback function
xhttp.onload = function() {
  // Here you can use the Data
}

// Send a request
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

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.

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**
onload	Defines a function to be called when the request is recieved (loaded)
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
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onload Property

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

The function is defined in the onload property of the XMLHttpRequest object:

Example

```
xhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

Multiple Callback Functions

If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task.

The function call should contain the URL and what function to call when the response is ready.

Example

```
loadDoc("url-1", myFunction1);

loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {cFunction(this);}
  xhttp.open("GET", url);
  xhttp.send();
}

function myFunction1(xhttp) {
  // action goes here
}
function myFunction2(xhttp) {
  // action goes here
}
```

The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a callback function to be executed when the readyState changes.

The status property and the statusText properties hold the status of the XMLHttpRequest object.

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

status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.

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

Example

```
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
```

The onreadystatechange event is triggered four times (1-4), one time for each change in the readyState.

AJAX - XMLHttpRequest

The XMLHttpRequest object is used to request data from 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)

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 default value for the async parameter is async = true.

You can safely remove the third parameter from your code.

Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

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:

Example

```
xhttp.open("GET", "demo_get.asp");
xhttp.send();
```

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

Example

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random());
xhttp.send();
```

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

Example

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

How the server uses the input and how the server responds to a request, is explained in a later chapter.

POST Requests

A simple POST request:

Example

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

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

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;
```

Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

Modern developer tools are encouraged to warn about using synchronous requests and may throw an InvalidAccessError exception when it occurs.

AJAX - Server Response

Server Response Properties

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

The responseText Property

The responseText property returns the server response as a JavaScript string, and you can use it accordingly:

Example

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The responseXML Property

The XMLHttpRequest object has an in-built XML parser.

The responseXML property returns the server response as an XML DOM object.

Using this property you can parse the response as an XML DOM object:

Example

Request the file cd_catalog.xml and parse the response:

```
const xmlDoc = xhttp.responseXML;
const x = xmlDoc.getElementsByTagName("ARTIST");

let txt = "";
for (let i = 0; i < x.length; i++) {
  txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;
```

```
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
```

Server Response Methods

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The getAllResponseHeaders() Method

The getAllResponseHeaders() method returns all header information from the server response.

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getAllResponseHeaders();
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

The getResponseHeader() Method

The getResponseHeader() method returns specific header information from the server response.

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getResponseHeader("Last-Modified");
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

AJAX XML Example

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

Get CD info

Example Explained

When a user clicks on the "Get CD info" button above, the loadDoc() function is executed.

The loadDoc() function creates an XMLHttpRequest object, adds the function to be executed when the server response is ready, and sends the request off to the server.

When the server response is ready, an HTML table is built, nodes (elements) are extracted from the XML file, and it finally updates the element "demo" with the HTML table filled with XML data:

```
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {myFunction(this);}
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("CD");
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
```

The XML File

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
```

```
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Greatest Hits</TITLE>
<ARTIST>Dolly Parton</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
<CD>
<TITLE>Still got the blues</TITLE>
<ARTIST>Gary Moore</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Virgin records</COMPANY>
<PRICE>10.20</PRICE>
<YEAR>1990</YEAR>
</CD>
</CATALOG>
```

XML Applications

This chapter demonstrates some HTML applications using XML, HTTP, DOM, and JavaScript.

The XML Document Used

In this chapter we will use the XML file called "cd_catalog.xml".

Display XML Data in an HTML Table

This example loops through each <CD> element, and displays the values of the <ARTIST> and the <TITLE> elements in an HTML table:

Example

```
<table id="demo"></table>

<script>
function loadXMLDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    const xmlDoc = xhttp.responseXML;
    const cd = xmlDoc.getElementsByTagName("CD");
    myFunction(cd);
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}

function myFunction(cd) {
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i < cd.length; i++) {
    table += "<tr><td>" +
    cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Display the First CD in an HTML div Element

This example uses a function to display the first CD element in an HTML element with id="showCD":

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  const xmlDoc = xhttp.responseXML;
  const cd = xmlDoc.getElementsByTagName("CD");
  myFunction(cd, 0);
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();

function myFunction(cd, i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

Navigate Between the CDs

To navigate between the CDs in the example above, create a next() and previous() function:

Example

```
function next() {
  // display the next CD, unless you are on the last CD
  if (i < len-1) {
    i++;
    displayCD(i);
  }
}

function previous() {
  // display the previous CD, unless you are on the first CD
  if (i > 0) {
    i--;
    displayCD(i);
  }
}
```

Show Album Information When Clicking On a CD

The last example shows how you can show album information when the user clicks on a CD:

Example

```
function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```