AJAX

1. Introduction

- Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser.
- **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.**
- This makes use of a browser's multithreaded design and lets one thread handle the browser and interactions while other threads wait for responses to asynchronous requests.
- **Responses to asynchronous requests are caught in JavaScript as events.**
- The events can subsequently trigger changes in the user interface or make additional requests. This differs from the typical synchronous requests which require the entire web page to refresh in response to a request.
- 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)

2. How AJAX Works

The following steps which are shown in Figure 1 describe how AJAX Works:

1. An event occurs in a web page (the page is loaded or 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

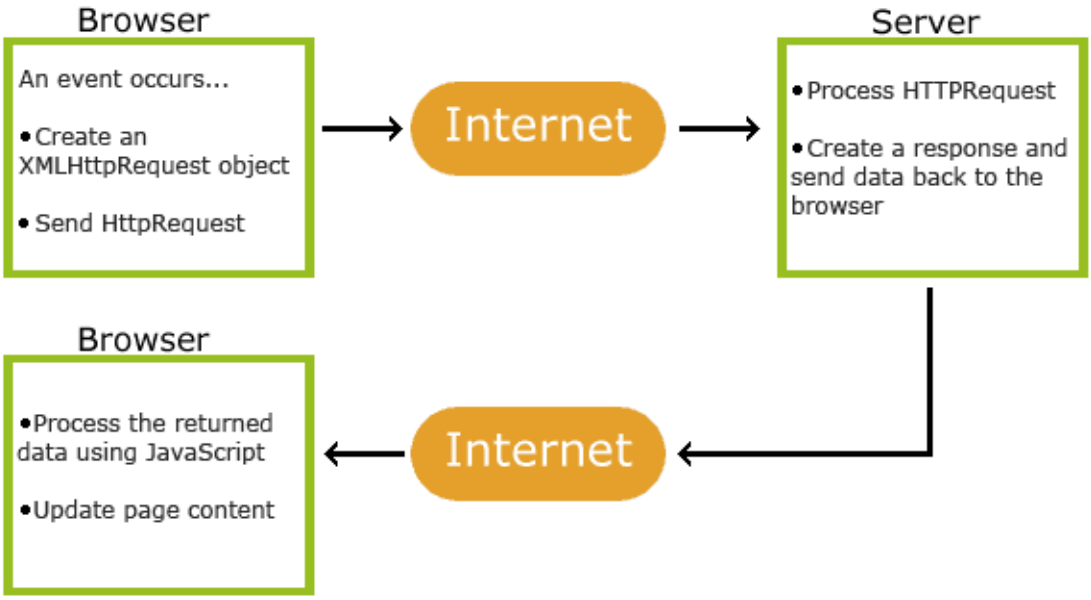


Figure 1. How AJAX works.

3. 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.

- The following steps are taken sequentially to achieve AJAX using 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

4.1 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();
```

4.2 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
}
```

4.3 Send a Request

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

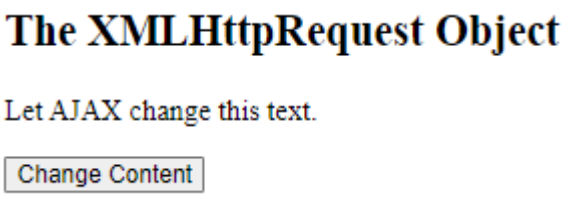
```
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>The XMLHttpRequest Object</h2>

<div id="demo">
<p>Let AJAX change this text.</p>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>

</body>
</html>
```

Output:

Before clicking "Change Content" button



After clicking "Change Content" button

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

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")

4. 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.

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();
```