AJAX

Definition of: AJaX engine
The code for an AJAX application. The use of the term "engine" is a misnomer, as it is not a standard module used by all developers. It is the JavaScript code written by the programmer.

Document Object Model (**DOM**) is a platform- and language-independent standard object model for representing HTML or XML and related formats. Strictly speaking, one should refer to "the DOM", but in practice, the definite article is usually dropped.

- The DOM accessed with a client-side scripting language, especially ECMAScript implementations such as JavaScript and JScript, to dynamically display and interact with the information presented.
- The XMLHttpRequest object is used to exchange data asynchronously with the web server. In some Ajax frameworks and in certain situations, an IFrame object is used instead of the XMLHttpRequest object to exchange data with the web server, and in other implementations, dynamically added <script> tags may be used.
- XML is sometimes used as the format for transferring data between the server and client, although any format will work, including preformatted HTML, plain text, JSON and even EBML. These files may be created dynamically by some form of server-side scripting.

Advantages and disadvantages

Advantages

1- User interface
The most obvious reason for using Ajax is an improvement to the user experience. Pages using Ajax behave more like a standalone application than a typical web page. Clicking on links that cause the entire page to refresh feels like a "heavy" operation. With Ajax, the page often can be updated dynamically, allowing a faster response to the user's interaction. While the full potential of Ajax has yet to be determined, some believe it will prove to be an important technology, helping make the Web even more interactive and popular than it currently is.

2- Bandwidth usage
By generating the HTML locally within the browser, and only bringing down JavaScript calls and the actual data, Ajax web pages can appear to load relatively quickly since the payload coming down is much smaller in size. An example of this technique is a large result set where multiple pages of data exist. With Ajax, the HTML of the page, e.g., a table structure with related TD and TR tags can be produced locally in the browser and not brought down with the first page of the document.

In addition to "load on demand" of contents, some web based applications load stubs of event handlers and then load the functions on the fly. This technique significantly cuts down the bandwidth consumption for web applications that have complex logic and functionality.

3- Separation of data, format, style, and function
A less specific benefit of the Ajax approach is that it tends to encourage programmers to clearly separate the methods and formats used for the different aspects of information delivery via the web. Although Ajax can appear to be a jumble of languages and techniques, and programmers are free to adopt and adapt whatever works for them, they are generally propelled by the development motive

itself to adopt separation between the following:

- Adopt separation between the *raw data or content to be delivered* - which is normally embedded in XML and sometimes derived from a server-side database.
- Adopt separation between the *format or structure of the webpage* - which is almost always built in HTML (or better, XHTML) and is then reflected and made available to dynamic manipulation in the DOM.
- Adopt separation between the *style elements of the webpage*: everything from fonts to picture placement are derived by reference to embedded or referenced CSS.
- Adopt separation between the *functionality of the web page* which is provided by a combination of
  - Javascript on the client browser (also called DHTML),
- Standard HTTP and XMLHttp for client-to-server communication, and
- Server-side scripting and/or programs using any suitable language preferred by the programmer to receive the client's specific requests and respond appropriately.

Disadvantages

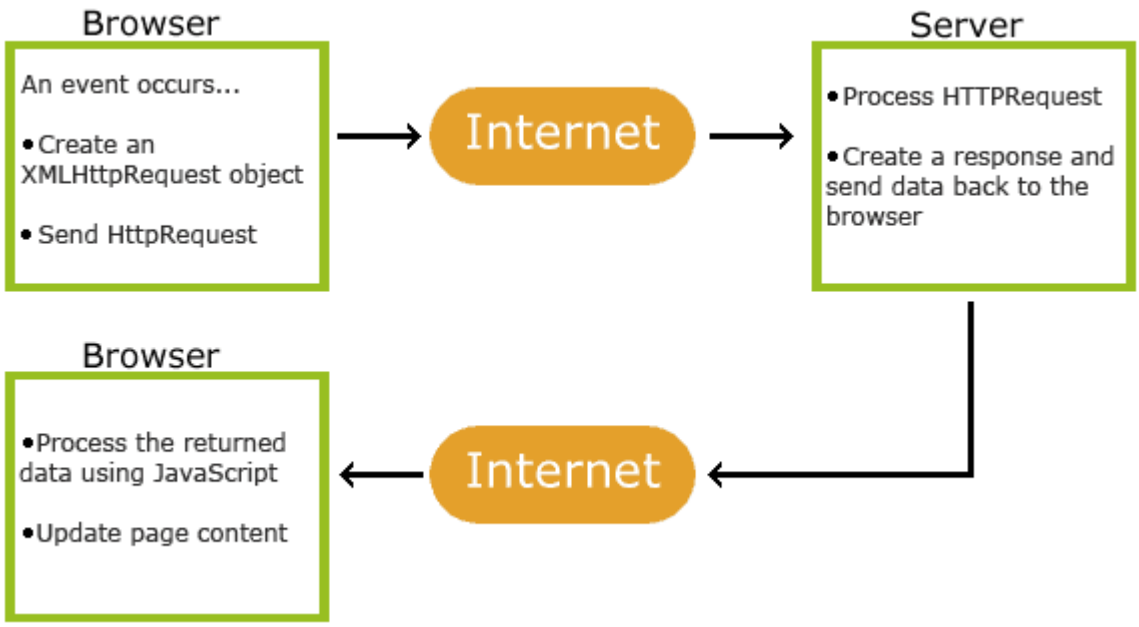
1- Browser integration

The dynamically created page does not register itself with the browser history engine, so triggering the "Back" function of the users' browser might not bring the desired result.

Developers have implemented various solutions to this problem. These solutions can involve using invisible IFRAMEs to invoke changes that populate the history used by a browser's back button. Google Maps, for example, performs searches in an invisible IFRAME and then pulls results back into an element on the visible web page. The World Wide Web Consortium (W3C) did not include an *iframe* element in its XHTML 1.1 Recommendation; the Consortium recommends the *object* element instead.

Another issue is that dynamic web page updates make it difficult for a user to bookmark a particular state of the application. Solutions to this problem exist, many of which use the URL

How AJAX Works



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

A simple GET request:

Example

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

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

Example

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

POST Requests

A simple POST request:

Example

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