AJAX

Ajax (Asynchronous JavaScript and XML) is a method of building interactive applications for the Web that process user requests immediately. Ajax combines several programming tools including JavaScript, dynamic HTML (DHTML), Extensible Markup Language (XML), cascading style sheets (CSS), the Document Object Model (DOM), and the Microsoft object, XMLHttpRequest. Ajax allows content on Web pages to update immediately when a user performs an action, unlike an HTTP request, during which users must wait for a whole new page to load. For example, a weather forecasting site could display local conditions on one side of the page without delay after a user types in a city name.
`

Google Maps is one well-known application that uses Ajax. The interface allows the user to change views and manipulate the map in real time. Ajax applications do not require installation of a plug-in, but work directly with a Web browser. Because of the technique's reliance on XMLHttpRequest, early applications worked only with Microsoft's Internet Explorer browser, but most other browsers now support Ajax.

Applications created with Ajax use an engine that acts as an intermediary between a **user's browser and the serv**er from which it is requesting information. Instead of loading a traditional Web page, the user's browser loads the Ajax engine, which displays the page the user sees. The engine continues to run in the background, using JavaScript to communicate with the Web browser. User input or clicking on the page sends a JavaScript call to the Ajax engine, which can respond instantly in many cases. If the engine needs additional data, it requests it from the server, usually using XML, while it is simultaneously updating the page.

Ajax is not a proprietary technology or a packaged product. Web developers have been using JavaScript and XML in combination for several years. Jesse James Garrett of the consultancy firm Adaptive Path is credited with coining the name "Ajax" as a shorthand way to refer to the specific technologies involved in a current approach.

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 fragment identifier (the portion of a URL after the '#'[6][7]) to keep track of, and allow users to return to, the application in a given state. This is possible because many browsers allow JavaScript to update the fragment identifier of the URL dynamically, so that Ajax applications can maintain it as the user changes the application's state. This solution also improves back-button support. It is, however, not a complete solution.

2- Response-time concerns

Network latency — or the interval between user request and server response — needs to be considered carefully during Ajax development. Without clear feedback to the user,[8] smart preloading of data and proper handling of the XMLHttpRequest object, users might experience delay in the interface of the web application, something which they might not expect or understand. Additionally, when an entire page is rendered there is a brief moment of re-adjustment for the eye when the content changes. The lack of this re-adjustment with smaller portions of the screen changing makes the latency more apparent. The use of visual feedback (such as throbbers) to alert the user of background activity and/or preloading of content and data are often suggested solutions to these latency issues.

Microsoft's Ajax Extensions addresses this network latency issue by offering the UpdateProgress control which allows the developer to alert the user that the page is being updated.

3- Search engine optimization

Websites that use Ajax to load data which should be indexed by search engines must be careful to provide equivalent data at a public, linked URL and in a format that the search engine can read, as search engines do not generally execute the JavaScript code required for Ajax functionality. This problem is not specific to Ajax, as the same issue occurs with sites that provide dynamic data as a full-page refresh in response to, say, a form submit (the general problem is sometimes called the hidden web).

4- Reliance on JavaScript

Ajax relies on JavaScript, which may be implemented differently by different browsers or versions of a particular browser. Because of this, sites that use JavaScript may need to be tested in multiple browsers to check for compatibility issues. It's not uncommon to see a JavaScript code written twice, one part for IE, another part for Mozilla compatibles. (see also Cross-platform web design). The level of IDE support for JavaScript is exceptionally poor.

An issue also arises if the user has switched off JavaScript support in the browser, thus disabling the

functionality of the page. Implementing **Mutual Exclusion** in JavaScript may solve the problem.

Extra Drawbacks

- In pre-HTML5 browsers, pages dynamically created using successive Ajax requests did not automatically register themselves with the browser's history engine, so clicking the browser's "back" button may not have returned the browser to an earlier state of the Ajax-enabled page, but may have instead returned to the last full page visited before it. Such behavior—navigating between pages instead of navigating between page states—may be desirable, but if fine-grained tracking of page state is required then a pre-HTML5 workaround was to use invisible iframes to trigger changes in the browser's history. A workaround implemented by Ajax techniques is to change the URL fragment identifier (the part of a URL after the '#') when an Ajax-enabled page is accessed and monitor it for changes.[9][10] HTML5 provides an extensive API standard for working with the browser's history engine.[11]
- Dynamic web page updates also make it difficult to bookmark and return to a particular state of the application. Solutions to this problem exist, many of which again use the URL fragment identifier.[9][10] The solution provided by HTML5 for the above problem also applies for this.[11]
- Depending on the nature of the Ajax application, dynamic page updates may interfere disruptively with user interactions, especially if working on an unstable Internet connection. For instance, editing a search field may trigger a query to the server for search completions, but the user may not know that a search completion popup is forthcoming, and if the internet connection is slow, the popup list may show up at an inconvenient time, when the user has already proceeded to do something else.
- Because most web crawlers do not execute JavaScript code,[12] publicly indexable web applications should provide an alternative means of accessing the content that would normally be retrieved with Ajax, thereby allowing search engines to index it.
- Any user whose browser does not support JavaScript or `XMLHttpRequest`, or simply has this functionality disabled, will not be able to properly use pages which depend on Ajax. Devices such as smartphones and PDAs may not have support for the required technologies, though this is becoming less of an issue. The only way to let the user carry out functionality is to fall back to non-JavaScript methods. This can be achieved by making sure links and forms can be resolved properly and not relying solely on Ajax.[13]
- Similarly, some web applications which use Ajax are built in a way that cannot be read by screen-reading technologies, such as JAWS. The WAI-ARIA standards provide a way to provide hints in such a case.[14]
- Screen readers that are able to use Ajax may still not be able to properly read the dynamically generated content.[15]
- The same origin policy prevents some Ajax techniques from being used across domains,[6] although the W3C has a draft of the XMLHttpRequest object that would enable this functionality.[16] Methods exist to sidestep this security feature by using a special Cross Domain Communications channel embedded as an iframe within a page,[17] or by the use of JSONP.
- The asynchronous callback-style of programming required can lead to complex code that is hard to maintain, to debug[18] and to test.[19]

```
<html>
<head>
<title>Simple Ajax Example</title>
<script language="Javascript">
function xmlhttpPost(strURL) {
    var xmlHttpReq = false;
    var self = this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        self.xmlHttpReq = new XMLHttpRequest();
    }
    // IE
    else if (window.ActiveXObject) {
        self.xmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
    }
    self.xmlHttpReq.open('POST', strURL, true);
    self.xmlHttpReq.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    self.xmlHttpReq.onreadystatechange = function() {
        if (self.xmlHttpReq.readyState == 4) {
            updatepage(self.xmlHttpReq.responseText);
        }
    }
    self.xmlHttpReq.send(getquerystring());
}

function getquerystring() {
    var form     = document.forms['f1'];
    var word = form.word.value;
    qstr = 'w=' + escape(word);  // NOTE: no '?' before querystring
    return qstr;
}

function updatepage(str){
```

```
    document.getElementById("result").innerHTML = str;
}
</script>
</head>
<body>
<form name="f1">
  <p>word: <input name="word" type="text">
  <input value="Go" type="button" onclick='JavaScript:xmlhttpPost("/csds.jsp")'></p>
  <div id="result"></div>
</form>
</body>
</html>
```

The full list of readyState values is:

State	Description
0	*The request is not initialized*
1	*The request has been set up*
2	*The request has been sent*
3	*The request is in process*
4	*The request is complete*

In practice you almost never use any of them except for 4.

Some XMLHttpRequest implementations may let you see partially received responses in `responseText` when `readyState==3`, but this isn't universally supported and shouldn't be relied upon.