



Ajax - A Primer

white paper

By Steve Cornish
© ThoughtBreak 2006

Ajax – A Primer

What is Ajax?

Ajax is not a technology – it is a group of technologies being used together in a particular way.

You will see many people claim Ajax is an acronym, and that AJAX stands for “Asynchronous JavaScript And XML” – this is not actually true but the notion is close enough to stick.

Ajax is the coming together of JavaScript (actually, ECMAScript, but we will refer to JavaScript from here on in), CSS, DOM, XHTML and XML to provide a way to create richly functional web applications that more closely resemble desktop applications than traditional request/response/refresh web sites.

The key feature of Ajax is the use of JavaScript to make asynchronous (read: user doesn’t wait) calls to the server, then using the response to dynamically update the current page. This allows the user to continue interacting with the site whilst the update is being performed.

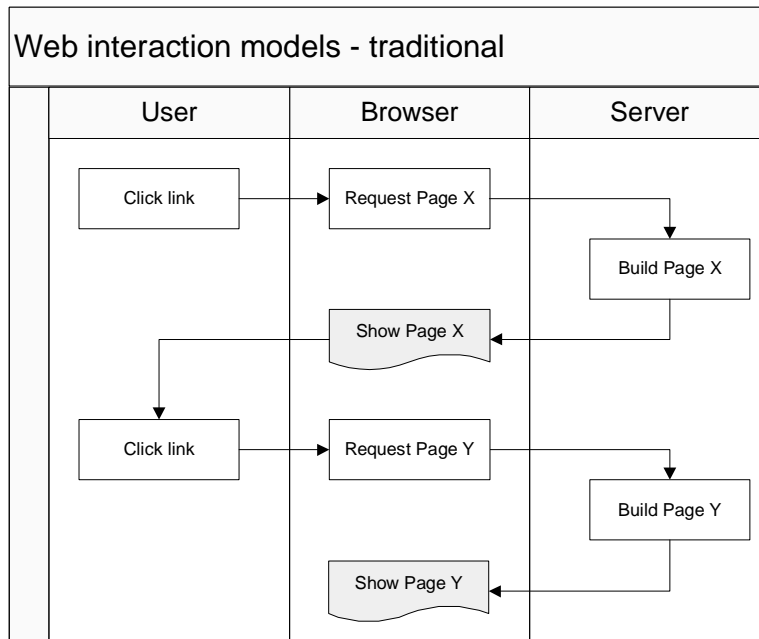
Ajax has become a reality because of modern browsers’ support of Web Standards:

- JavaScript
- CSS
- DOM
- XHTML
- XML

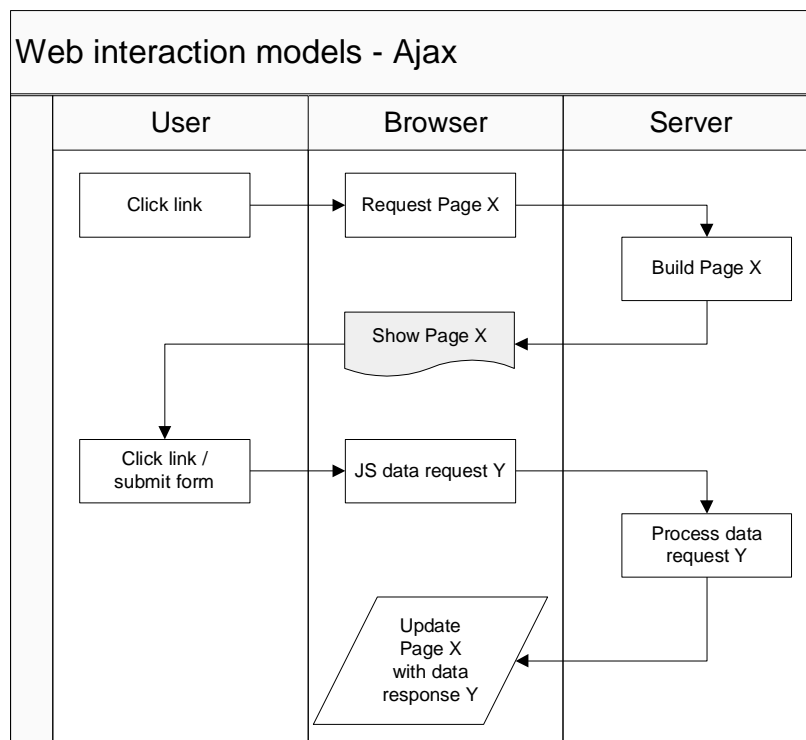
A “rich, interactive, user-friendly interface based on Ajax,” is considered one of the key characteristics of a Web 2.0 site (ref: Wikipedia: Web 2.0). The hype around Web 2.0 is certainly helping to bring Ajax into the mainstream.

How does it work?

The traditional model for web applications is request/response/refresh:



Here, you can see that two separate pages are built and returned by the server. But if the two pages are similar, rebuilding the entire page seems wasteful. Ajax solves this problem by allowing a “behind the scenes” request to be sent (using JavaScript), and allowing the current page to be partially updated (using DOM) without causing a refresh:



Key technologies used in Ajax

Let's consider the role of the constituent technologies in an Ajax application:

XHTML

XHTML should be used to deliver the *structure* of the page. Resist the urge to have any stylistic markup – this is the purpose of CSS.

CSS

The association of one or more stylesheets with an XHTML page provides the styling of the page. If the XHTML has successfully delivered structure without style, you should still have a readable and usable site even without a stylesheet attached.

When done correctly, the combination of XHTML and CSS can deliver a rich, complex web page with a surprisingly small amount of bytes.

Note: many sites now offer the user a choice of “skins” – these are essentially substituting one stylesheet for another, which demonstrates the separation of structure and style perfectly.

JavaScript

ECMAScript defined an XMLHttpRequest object that can be used to send requests (both synchronous and asynchronous) to URIs and to receive the response returned (by declaring which function to invoke when the response is received).

The response can be obtained as a JavaScript string, or as an XML document (if the response is XML).

The callback function invoked when the response is retrieved can update the current page by manipulating the page's DOM.

DOM

The page's Document Object Model is available to JavaScript. JavaScript can update the page by manipulating the page's DOM.

Also, should the Ajax response be XML, the response's DOM can be obtained from the XMLHttpRequest.

XML

XML is everywhere! Your page structure was written in XML (i.e. XHTML), which means you can easily modify it via the DOM.

Also the Ajax response can be XML. It does not have to be, though – it could be a single text value, a custom data format, or JSON (more on JSON later).

An example

This example is the web site of a pizza store.



Using Ajax, we want to lookup and populate the delivery address when the user enters a phone number.

The full XHTML and CSS is skipped here for brevity, but we need a trigger to start the lookup. This will be based on the **onChange** event of the phone number input field:

```
<p>Enter your phone number:
  <input type="text" size="14"
        name="phone" id="phone"
        onChange="getCustomerAddress();" />
</p>
```

Now we need to implement the `getCustomerAddress()` JavaScript function to obtain the address info from the phone number:

```
var request = createRequest(); // returns a XMLHttpRequest object
                          // impl skipped for brevity

function getCustomerAddress() {
  var phone = document.getElementById("phone").value;
  var url = "getCustomerAddress.php?phone=" + escape(phone);
  request.open("GET", url, true);
```

```
request.onreadystatechange = updatePage;
request.send(null);
}
```

Key lines in this function are:

```
var phone = document.getElementById("phone").value;
```

Obtains the value of the 'phone' field using the DOM.

```
request.open("GET", url, true);
```

Tells the XMLHttpRequest object what request to make. Here "true" means "make an asynch call". Note the request is not sent here.

```
request.onreadystatechange = updatePage;
```

Here, we tell the XMLHttpRequest object to call the updatePage() function whenever the *ready state* changes. Ready states are explained below.

```
request.send(null);
```

Here we send the request with no body. If we were sending a POST, we could attach the body here.

Ready states

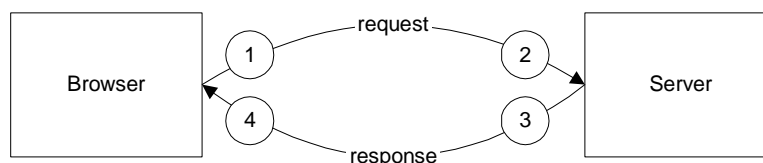
Above, the line:

```
request.onreadystatechange = updatePage;
```

... told the XMLHttpRequest object to call the updatePage() function whenever the ready state changes. The ready state is an indicator of how far through a request you are.

Ready states are:

- 0 = uninitialized
- 1 = open
- 2 = sent
- 3 = receiving
- 4 = loaded



Since we're only interested in the response once received, we are only interested in a ready state of 4.

So we know that the XMLHttpRequest object will call our callback function updatePage() whenever the ready state changes. We also

need to check the HTTP response code indicates a successful response (status = 200), so our callback function looks like:

```
function updatePage() {
  if (request.readyState == 4) {           // check ready state
    if (request.status == 200) {         // check HTTP status
      /* Get the response from the server */
      var customerAddress = request.responseText;

      /* Update the HTML web form */
      document.getElementById("address").value =
        customerAddress;
    } else {
      alert("Error! Request status is " + request.status);
    }
  }
}
```

In this example, the response received is plain text, and so we retrieved it using:

```
request.responseText
```

If the response was returned as XML, we could have obtained the DOM of the response by using:

```
request.responseXML
```

This is a very simple example of using Ajax, but it does identify the key participants clearly.

JSON – JavaScript Object Notation

Whilst XML seems a natural fit for this technology stack, on closer inspection using XML as the response format seems a little clunky.

JSON is a structured data notation:

```
{ "salesTotals": [
  {
    "wheelsSold": 20,
    "bearingsSold": 14,
    "boardsSold": 76
  }
]};
```

Here, "salesTotals" is at the top-level and contains three properties. These three properties are "wheelsSold", "bearingsSold" and "boardsSold", and have the values 20, 14 and 76 respectively.

The great thing about JSON is that with one wave of the wand, you can work with it using normal JavaScript notation. First, the wand:

```
var jsonResponse = eval( '(' + request.responseText + ')' );
```

Now the normal JavaScript:

```
var wheelsSold    = jsonResponse.salesTotals[0].wheelsSold;
var bearingsSold = jsonResponse.salesTotals[0].bearingsSold;
var boardsSold   = jsonResponse.salesTotals[0].boardsSold;
```

JSON *is* JavaScript – you do not need any third party code to work with it in your existing JavaScripts. However, the server-side may need a little help to format the JSON response string. Fortunately, there are JSON libraries available for pretty much any server-side language.

How ThoughtBreak uses Ajax

ThoughtBreak uses Ajax in many of our clients' web applications. Examples are:

- Performing behind-the-scenes loading of account details (from a backend billing system)
- Polling for a change of state of a customer's order (reading the workflow repository)
- Providing a no-page-refresh shopping basket for a rich, responsive experience

Accessibility

The Disability Discrimination Act (DDA) has direct relevance to public services offered over the web, and using Ajax in web applications offers challenges for organisations interested in complying with this law.

Developers building Ajax enabled sites need to consider alternative or fallback options for users running alternative browsers (e.g. voice) since Ajax typically relies on features present in graphical browsers.

For example, where Ajax may be used to update content in specific portions of a web page, non-Ajax users would ideally be able to load and manipulate the whole page as a fallback. This allows the developers to preserve the experience of users in non-Ajax environments whilst giving those with capable browsers a much more responsive experience.

Pros and Cons

- +Low bandwidth
- +Interactive, responsive applications
- Back button and bookmarking behaviour not as user expects (workarounds exist)
- Accessibility considerations

Resources

Ajax packages

There are plenty of Ajax packages out there, with more appearing every day. Two of the more popular and functionally complete packages are:

Script.aculo.us - <http://script.aculo.us/> (free)

Open RICO - <http://openrico.org/rico/home.page> (open source)

Both of these packages are built on top of the Prototype JavaScript Framework:

<http://prototype.conio.net/>

Further reading

Ajax: A new approach to web applications (the essay that kicked it all off) -

<http://www.adaptivepath.com/publications/essays/archives/000385.php>

Disability Discrimination Act and Web Accessibility -

<http://www.webcredible.co.uk/user-friendly-resources/web-accessibility/uk-website-legal-requirements.shtml>

Wikipedia: JSON – JavaScript Object Notation -

http://en.wikipedia.org/wiki/JavaScript_Object_Notation

JSON.org – for server-side JSON libraries - <http://www.json.org/>