AJAX: Getting Started

From MDC

This article guides you through the AJAX basics and gives you two simple hands-on examples to get you started.

Contents

[hide]

1 What's AJAX?
2 Step 1 – say "Please!" or How to Make an HTTP Request
3 Step 2 – "There you go!" or Handling the Server Response
4 Step 3 – "All together now!" - A Simple Example
5 Step 4 – "The X-Files" or Working with the XML Response

What's AJAX?

AJAX (Asynchronous JavaScript and XML) is a newly coined term for two powerful browser features that have been around for years, but were overlooked by many web developers until recently when applications such as Gmail, Google suggest, and Google Maps hit the streets.

The two features in question are that you can:

- Make requests to the server without reloading the page
- Parse and work with XML documents

Step 1 – say "Please!" or How to Make an HTTP Request

In order to make an HTTP request to the server using JavaScript, you need an instance of a class that provides you this functionality. Such a class was originally introduced in Internet Explorer as an ActiveX object, called XMLHttpRequest. Then Mozilla, Safari and other browsers followed, implementing an XMLHttpRequest class that supports the methods and properties of Microsoft's original ActiveX object.

As a result, in order to create a cross-browser instance (object) of the required class, you can do:
Some versions of some Mozilla browsers won't work properly if the response from the server doesn't have an XML mime-type header. To satisfy this, you can use an extra method call to override the header sent by the server, just in case it's not `text/xml`.

```
http_request.overrideMimeType('text/xml');
```

The next thing is to decide what you want to do after you receive the server response to your request. At this stage you just need to tell the HTTP request object which JavaScript function will do the work of processing the response. This is done by setting the `onreadystatechange` property of the object to the name of the JavaScript function you plan to use, like this:

```
http_request.onreadystatechange = nameOfTheFunction;
```

Note that there are no brackets after the function name and no parameters passed. Also, instead of giving a function name, you can use the Javascript technique of defining functions on the fly and define the actions that will process the response right away, like this:

```
http_request.onreadystatechange = function(){
    // do the thing
};
```

Next, after you've declared what will happen as soon as you receive the response, you need to actually make the request. You need to call the `open()` and `send()` methods of the HTTP request class, like this:

```
http_request.open('GET', 'http://www.example.org/some.file', true);
http_request.send(null);
```

- The first parameter of the call to `open()` is the HTTP request method – GET, POST, HEAD or any other method you want to use and that is supported by your server. Keep the method capitalized as per the HTTP standard; otherwise some browsers (like Firefox) might not process the request. For more information on the possible...
HTTP request methods you can check the [W3C specs](http://www.w3.org/褐色). 

- The second parameter is the URL of the page you’re requesting. As a security feature, you cannot call pages on 3rd-party domains. Be sure to use the exact domain name on all of your pages or you will get a 'permission denied' error when you call open(). A common pitfall is accessing your site by domain.tld, but attempting to call pages with www.domain.tld.
- The third parameter sets whether the request is asynchronous. If `true`, the execution of the JavaScript function will continue while the response of the server has not yet arrived. This is the A in AJAX.

The parameter to the `send()` method can be any data you want to send to the server if POST-ing the request. The data should be in the form of a query string, like:

```
name=value&anothername=othervalue&so=on
```

Note that if you want to POST data, you have to change the MIME type of the request using the following line:

```
http_request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
```

Otherwise, the server will discard the POSTed data.

### Step 2 – "There you go!" or Handling the Server Response

Remember that when you were sending the request, you provided the name of a JavaScript function that is designed to handle the response.

```
http_request.onreadystatechange = nameOfTheFunction;
```

Let's see what this function should do. First, the function needs to check for the state of the request. If the state has the value of 4, that means that the full server response is received and it's OK for you to continue processing it.

```
if (http_request.readyState == 4) {
    // everything is good, the response is received
} else {
    // still not ready
}
```

The full list of the `readyState` values is as follows:

- 0 (uninitialized)
- 1 (loading)
- 2 (loaded)
- 3 (interactive)
The next thing to check is the status code of the HTTP server response. All the possible codes are listed on the W3C site. For our purposes we are only interested in 200 OK response.

```javascript
if (http_request.status == 200) {
    // perfect!
} else {
    // there was a problem with the request,
    // for example the response may be a 404 (Not Found)
    // or 500 (Internal Server Error) response codes
}
```

Now after you've checked the state of the request and the HTTP status code of the response, it's up to you to do whatever you want with the data the server has sent to you. You have two options to access that data:

- `http_request.responseText` – will return the server response as a string of text
- `http_request.responseXML` – will return the response as an `XMLDocument` object you can traverse using the JavaScript DOM functions

Step 3 – "All together now!" - A Simple Example

Let's put it all together and do a simple HTTP request. Our JavaScript will request an HTML document, `test.html`, which contains the text "I'm a test." and then we'll `alert()` the contents of the `test.html` file.
In this example:

- The user clicks the link "Make a request" in the browser;
- This calls the `makeRequest()` function with a parameter - the name `test.html` of an HTML file in the same directory;
- The request is made and then (onreadystatechange) the execution is passed to
alertContents();

- alertContents() checks if the response was received and it's an OK and then
  alert()s the contents of the test.html file.

You can test the example here and you can see the test file here.

**Note:** The line http_request.overrideMimeType('text/xml'); above will cause Javascript Console errors in Firefox 1.5b as documented at https://bugzilla.mozilla.org/show_bug.cgi?id=311724 if the page called by XMLHttpRequest is not valid XML (eg, if it is plain text).

If you get Syntax Error or Not Well Formed Error on that browser, and you're not trying to load an XML page from XMLHttpRequest, remove that line from your code.

**Note 2:** if you are sending a request to a piece of code that will return XML, rather than to a static XML file, you must set some response headers if your page is to work in Internet Explorer in addition to Mozilla. If you do not set header Content-Type: application/xml, IE will throw a Javascript error 'Object Expected' after the line where you try to access an XML element. If you do not set header Cache-Control: no-cache the browser will cache the response and never re-submit the request, making debugging 'challenging'.

### Step 4 – "The X-Files" or Working with the XML Response

In the previous example, after the response to the HTTP request was received, we used the `responseText` property of the request object and it contained the contents of the test.html file. Now let's try the `responseXML` property.

First off, let's create a valid XML document that we'll request later on. The document (test.xml) contains the following:

```xml
<?xml version="1.0" ?>
<root>
    I'm a test.
</root>
```

In the script we only need to change the request line with:

```javascript
... onclick="makeRequest('test.xml')"> ...
```

Then in `alertContents()` we need to replace the `alert()`-ing line `alert(http_request.responseText);` with:
This way we took the XMLDocument object given by responseXML and we used DOM methods to access some data contained in the XML document. You can see the test.xml here and the updated test script here.

For more on the DOM methods, be sure to check Mozilla's DOM implementation documents.