Getting Started

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

What's AJAX?

AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML. In a nutshell, it is the use of the `XMLHttpRequest` object to communicate with servers. It can send and receive information in various formats, including JSON, XML, HTML, and text files. AJAX's most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh the page.

The two major features of AJAX allow you to do the following:

- Make requests to the server without reloading the page
- Receive and work with data from the server

Step 1 – How to make an HTTP request

In order to make an HTTP request to the server with JavaScript, you need an instance of an object with the necessary functionality. This is where `XMLHttpRequest` comes in. Its predecessor originated in Internet Explorer as an ActiveX object called `XMLHTTP`. Then, Mozilla, Safari, and other browsers followed, implementing an `XMLHttpRequest` object that supported the methods and properties of Microsoft's original ActiveX object. Meanwhile, Microsoft implemented XMLHttpRequest as well.

```
// Old compatibility code, no longer needed.
if (window.XMLHttpRequest) { // Mozilla, Safari, IE7+ ...
```

```
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 and older
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Note: For illustration purposes, the above is a somewhat simplified version of the code to be used for creating an XMLHttp instance. For a more realistic example, see step 3 of this article.

After making a request, you will receive a response back. At this stage, you need to tell the XMLHttp request object which JavaScript function will handle the response, by setting the `onreadystatechange` property of the object and naming it after the function to call when the request changes state, like this:

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

Note that there are no parentheses or parameters after the function name, because you're assigning a reference to the function, rather than actually calling it. Alternatively, instead of giving a function name, you can use the JavaScript technique of defining functions on the fly (called "anonymous functions") to define the actions that will process the response, like this:

```
httpRequest.onreadystatechange = function(){
    // Process the server response here.
};
```

Next, after declaring what happens when you receive the response, you need to actually make the request, by calling the `open()` and `send()` methods of the HTTP request object, like this:

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

- The first parameter of the call to `open()` is the HTTP request method – GET, POST, HEAD, or another method supported by your server. Keep the method all-capitals as per the HTTP standard, otherwise some browsers (like Firefox) might not process the request. For more information on the possible HTTP request methods, check the W3C specs.

- The second parameter is the URL you're sending the request to. As a security feature, you cannot call URLs on 3rd-party domains by default. 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`. If you really need to send a request to another domain, see HTTP access control (CORS).
- The optional third parameter sets whether the request is asynchronous. If `true` (the default), JavaScript execution will continue and the user can interact with the page while the server response has yet to arrive. This is the first 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. Form data should be sent in a format that the server can parse, like a query string:

```
"name=value&anothername="+encodeURIComponent(myVar)+"&so=on"
```

or other formats, like `multipart/form-data`, JSON, XML, and so on.

Note that if you want to `POST` data, you may have to set the MIME type of the request. For example, use the following before calling `send()` for form data sent as a query string:

```
httpRequest.setRequestHeader('Content-Type', 'application/x-www-fc
```

Step 2 – Handling the server response

When you sent the request, you provided the name of a JavaScript function to handle the response:

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

What should this function do? First, the function needs to check the request's state. If the state has the value of `XMLHttpRequest.DONE` (corresponding to 4), that means that the full server response was received and it's OK for you to continue processing it.

```
if (httpRequest.readyState === XMLHttpRequest.DONE) {
    // Everything is good, the response was received.
} else {
    // Not ready yet.
}
```

The full list of the `readyState` values is documented at XMLHTTPRequest.readyState and is as follows:

- 0 (uninitialized) or (**request not initialized**)
- 1 (loading) or (**server connection established**)
- 2 (loaded) or (**request received**)
- 3 (interactive) or (**processing request**)
- 4 (complete) or (**request finished and response is ready**)

Next, check the HTTP response status codes of the HTTP response. The possible codes are listed at the W3C. In the following example, we differentiate between a successful and unsuccessful AJAX call by checking for a `200 OK` response code.

```
if (httpRequest.status === 200) {
    // Perfect!
} else {
    // There was a problem with the request.
    // For example, the response may have a 404 (Not Found)
    // or 500 (Internal Server Error) response code.
}
```

After checking the state of the request and the HTTP status code of the response, you can do whatever you want with the data the server sent. You have two options to access that data:

- `httpRequest.responseText` – returns the server response as a string of text
- `httpRequest.responseXML` – returns the response as an `XMLDocument` object you can traverse with JavaScript DOM functions

Note that the steps above are valid only if you used an asynchronous request (the third parameter of `open()` was unspecified or set to `true`). If you used a **synchronous** request you don't need to specify a function, but this is highly discouraged as it makes for an awful user experience.

Step 3 – A Simple Example

Let's put it all together with a simple HTTP request. Our JavaScript will request an HTML document, `test.html`, which contains the text "I'm a test." Then we'll `alert()` the contents of the response. Note that this example uses vanilla JavaScript — no jQuery is involved. Also, the HTML, XML and PHP files should be placed in the same directory.

```
<button id="ajaxButton" type="button">Make a request</button>

<script>
(function() {
  var httpRequest;
  document.getElementById("ajaxButton").addEventListener('click',

  function makeRequest() {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
      alert('Giving up :( Cannot create an XMLHTTP instance');
      return false;
    }
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', 'test.html');
    httpRequest.send();
  }

  function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText);
      } else {
        alert('There was a problem with the request.');
      }
    }
  }
})();
</script>
```

In this example:

- The user clicks the "Make a request" button;
- The event handler calls the `makeRequest()` function;
- The request is made and then (`onreadystatechange`) the execution is passed to `alertContents()`;
- `alertContents()` checks if the response was received and OK, then `alert()`s the contents of the `test.html` file.

> **Note**: If you're sending a request to a piece of code that will return XML, rather than a static HTML file, you must set response headers to work in Internet Explorer. If you do not set header `Content-Type: application/xml`, IE will throw a JavaScript "Object Expected" error after the line where you tried to access an XML element.

> **Note 2**: 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. You can also add an always-different GET parameter, like a timestamp or random number (see bypassing the cache)

> **Note 3**: If the `httpRequest` variable is used globally, competing functions calling `makeRequest()` can overwrite each other, causing a race condition. Declaring the `httpRequest` variable local to a closure containing the AJAX functions avoids this.

In the event of a communication error (such as the server going down), an exception will be thrown in the `onreadystatechange` method when accessing the response status. To mitigate this problem, you could wrap your `if...then` statement in a `try...catch`:

```
function alertContents() {
  try {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText);
      } else {
        alert('There was a problem with the request.');
      }
    }
  }
  catch( e ) {
    alert('Caught Exception: ' + e.description);
  }
}
```

Step 4 – Working with the XML response

In the previous example, after receiving the response to the HTTP request we used the request object's `responseText` property , which 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 version="1.0" ?>
<root>
    I'm a test.
</root>
```

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

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

Then in `alertContents()`, we need to replace the line `alert(httpRequest.responseText);` with:

```
var xmldoc = httpRequest.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
alert(root_node.firstChild.data);
```

This code takes the `XMLDocument` object given by `responseXML` and uses DOM methods to access some of the data contained in the XML document. You can see the `test.xml` here and the updated test script here.

Step 5 – Working with data

Finally, let's send some data to the server and receive a response. Our JavaScript will request a dynamic page this time, `test.php`, which will take the data we send and return a "computed" string - "Hello, [user data]!" - which we'll `alert()`.

First we'll add a text box to our HTML so the user can enter their name:

```
<label>Your name:
  <input type="text" id="ajaxTextbox" />
</label>
<span id="ajaxButton" style="cursor: pointer; text-decoration: unc
  Make a request
</span>
```

We'll also add a line to our event handler to get the user's data from the text box and send it to the `makeRequest()` function along with the URL of our server-side script:

```
  document.getElementById("ajaxButton").onclick = function() {
      var userName = document.getElementById("ajaxTextbox").value;
      makeRequest('test.php',userName);
  };
```

We need to modify `makeRequest()` to accept the user data and pass it along to the server. We'll change the request method from `GET` to `POST`, and include our data as a parameter in the call to `httpRequest.send()`:

```
  function makeRequest(url, userName) {

    ...

    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('POST', url);
    httpRequest.setRequestHeader('Content-Type', 'application/x-ww
    httpRequest.send('userName=' + encodeURIComponent(userName));
  }
```

The function `alertContents()` can be written the same way it was in Step 3 to alert our computed string, if that's all the server returns. However, let's say the server is going to return both the computed string and the original user data. So if our user typed "Jane" in the text box, the server's response would look like this:

```
{"userData":"Jane","computedString":"Hi, Jane!"}
```

To use this data within `alertContents()`, we can't just alert the `responseText`, we have to parse it and alert `computedString`, the property we want:

```
function alertContents() {
  if (httpRequest.readyState === XMLHttpRequest.DONE) {
    if (httpRequest.status === 200) {
      var response = JSON.parse(httpRequest.responseText);
      alert(response.computedString);
    } else {
      alert('There was a problem with the request.');
    }
  }
}
```

The `test.php` file should contain the following:

```
$name = (isset($_POST['userName'])) ? $_POST['userName'] : 'no name';
$computedString = "Hi, " . $name . "!";
$array = ['userName' => $name, 'computedString' => $computedString];
echo json_encode($array);
```

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

Last modified: Mar 18, 2019, by MDN contributors