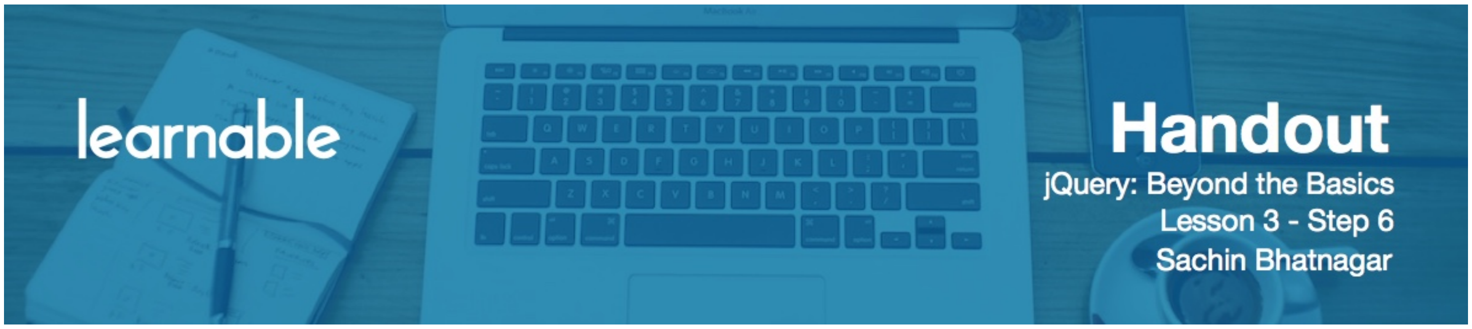

Handout

jQuery: Beyond the Basics
Lesson 3 - Step 6
Sachin Bhatnagar

Introduction to jQuery $.ajax method

So far all the AJAX methods that we have seen in this lesson were shorthands. If you want full-fledged control, however, the jQuery offers the `$.ajax` method:

```
$.ajax({
    url: "/api/serve.php",
    type: "GET",
    data: somedata,
    success: function(response) {},
    error: function() {}
});
```

This method accepts a JavaScript object that lets you define a lot of various parameters.

Planning file upload to server using AJAX

Open the *uploadFiles_Begin.htm* file before proceeding.

We have a simple user interface with four components:

- An input field that lets you browse and pick a file to upload.
- An "Upload now" button to submit the form.
- An HTML5 progress bar element to visually display uploading progress.
- An unordered list in the lower section of the UI displays the contents of the *uploads* folder on the server.

We're once again using a PHP script on the backend that receives the file on the server and places it in the *uploads* folder.

Implement file upload with $.ajax method

Please note that if you're on Linux, the *uploads* folder must be writeable.

I've already created a `listFiles` function that should be called once the file upload completes. It uses the `$.getJSON` method to retrieve and display the contents of the *upload* folder on the server. This data is returned as JSON by the *upload.php* file, which we are also using to receive the file on the server.

Note that the maximum size of a file that PHP can accept through a POST request is set to 8 MB by default on most server configurations. You can change this by editing your *php.ini* file (look for the `post_max_size` parameter).

```
$("#uploadNow").on("click", function(evt) {
  var file_to_upload = uploadFile[0].files[0];
  if(file_to_upload != undefined) {
// Upload file using Ajax
    var packFile = new FormData();
  }
});
```

We are creating a variable called `packFile` and set it to a new instance of the `FormData` API. This API lets us programmatically generate key-value pairs that represent form fields and values with the form encoding set to `multipart/form-data`. Do note that Internet Explorer only supports the FormData API in versions 10 and above.

```
$("#uploadNow").on("click", function(evt) {
  var file_to_upload = uploadFile[0].files[0];
  if (file_to_upload != undefined) {
// Upload file using Ajax
```

```
    var packFile = new FormData();
    packFile.append("file", file_to_upload);
    $.ajax({
      url: uploadURI,
      type: 'POST',
      data: packFile,
      processData: false,
      contentType: false,
      success: function (data) {
        progressBar.val(0);
        listFiles();
      }
    });
  }
});
```

`packfile.append` sets the file to upload.

For the actual AJAX call, `processData` option is set to `false` to prevent the `$.ajax` method from converting the data being sent to a query string. `contentType` is set to `false` to prevent setting any content type headers since that is already preset in our FormData object.

Inside the `success` event handler function we simply call the `listFiles` function which will fetch and update our unordered list on the UI that displays the contents of the *uploads* folder on the server.

We have not yet animated the progress bar element. When using the `XMLHttpRequest` object we can listen to the progress event on the upload property and fetch the amount of data that has been uploaded so far. The `$.ajax` method doesn't give us direct access to this event, but it does allow to modify the `XMLHttpRequest` object that it uses internally.

Customize XMLHttpRequest object

```
$("#uploadNow").on("click", function(evt) {
  var file_to_upload = uploadFile[0].files[0];
  if (file_to_upload != undefined) {
// Upload file using Ajax
    var packFile = new FormData();
    packFile.append("file", file_to_upload);
    $.ajax({
      xhr: function () {
        var xhr = new XMLHttpRequest();
        xhr.upload.addEventListener("progress", function (evt) {
          if (evt.lengthComputable) {
            var percentComplete = evt.loaded / evt.total;
            progressBar.val(Math.round(percentComplete * 100));
          }
        }, false);
        return xhr;
      },
      url: uploadURI,
      type: 'POST',
      data: packFile,
      processData: false,
      contentType: false,
      success: function (data) {
        progressBar.val(0);
        listFiles();
      }
    });
  }
});
```

We've created the progress event listener function on the `xhr.upload` property. Inside this function we check to see if the `evt.lengthComputable` property is available or not. This property, if set to `true`, means that we can quantify the amount of data that is being sent. So we can calculate the amount of data that has been uploaded versus the total amount of data that needs to be uploaded. This will be a value between 0 and 1, but we are multiplying this by 100 to match up to the 0 to 100 scale of our progress bar, and round off this value.

Please note that you have to return the `xhr` object to the `$.ajax` method.

Now you may check how this all is working in your browser.