Upload Multiple Files Asynchronously - Part 1

In this blog post, you are going to learn how to use jQuery, JavaScript, Ajax, and a Web API method to upload multiple files asynchronously. As each file is uploaded, a progress bar is displayed to indicate the progress for each file, as shown in Figure 1. In order to accomplish this, you learn to clone an HTML template for each file selected to upload. Yes, you can find free, open-source libraries to help you do this, but it is always good to know how these things work under the hood.

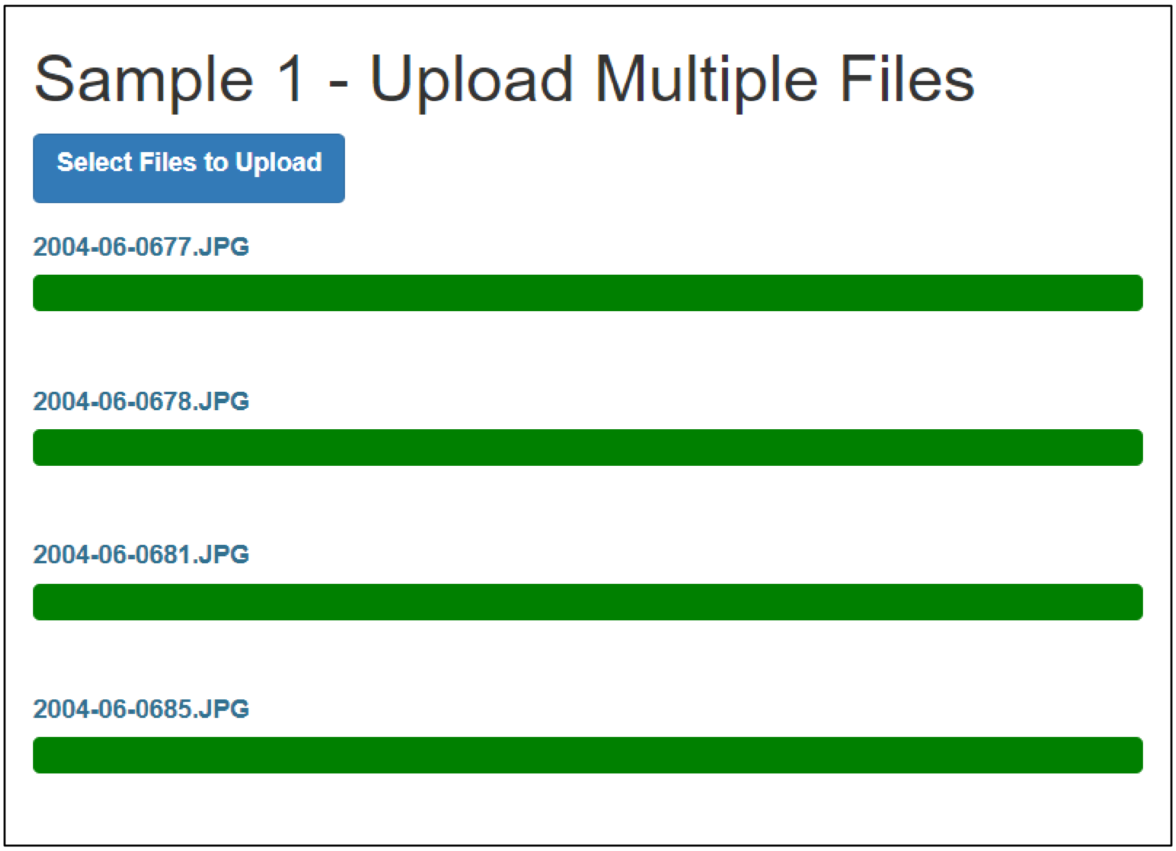


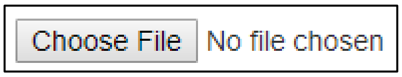
Figure 1: Provide feedback to the user when uploading multiple files.

Working with the File Input Element

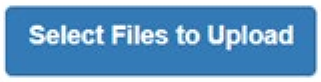
When you add an input element and set the type attribute set to "file", you may add the attribute “multiple” to allow the user to select more than one file at a time for uploading.

```
<input type="file" multiple="multiple" />
```

When you use the file input element, your browser may display this input like the following. However, the look and feel will be different on each browser.



If you use the Bootstrap CSS framework, you can make this input look consistent across browsers, as shown below.



For more information on how to style the file input element, read my previous blog post at https://bit.ly/2ZWv4Ff.

Create HTML Page for Uploading Files

To build this sample, I am going to use Visual Studio, jQuery, Bootstrap 3.x and the Microsoft ASP.NET Web API. However, feel free to use whatever tools you want. All the front-end code is generic and can be used with any JavaScript framework. While I am using Microsoft's ASP.NET Web API, you should be able to adapt the code to any Web API framework with ease.

To start creating this sample, bring up Visual Studio and select ASP.NET Web Application (.NET Framework) project. Choose the **Empty** template but select the **Web API** option. Using the **NuGet Package Manager**, add Bootstrap 3.4.x to your project. Make sure you are installing Bootstrap 3.4.x and not Bootstrap 4.x. Installing Bootstrap 3.4.x also installs jQuery. After installing these, click on the **Updates** tab in the NuGet Package Manager window and update any packages, except Bootstrap.

Add an index.html page in the root of your project. Locate the <head> element and modify it to look like the following code.

```
<head>
  <title>Upload Multiple Files Sample</title>
  <meta charset="utf-8" />
  <meta name="viewport"
        content="width=device-width, initial-scale=1" />

  <link href="/Content/bootstrap.min.css" rel="stylesheet" />
  <link href="/Content/site.css" rel="stylesheet" />
</head>
```

Right mouse-click on the **Content** folder and add a new style sheet named **site.css**. Add the following rule in the site.css file.

```
.progress-extra-space {
   margin-top: 1em;
}
```

Back in the index.html page, just below the <body> element add a <div class="container"> element. This is the main wrapper for Bootstrap to contain all other Bootstrap CSS classes.

```
<div class="container">
  <h1>Upload Multiple Files Sample</h1>

</div>
```

Add File Input Element

Just below the <h1> element, add another <div> with the *class* attribute set to "form-group btn btn-primary". These Bootstrap classes style the file input element to look like a Bootstrap button instead of a normal HTML button. Add a label and an input type within the div. Notice the input element has the *multiple* attribute set. In addition, it has the *style* attribute set to "display:none;". This allows the Bootstrap CSS classes to style the file input element. The *onchange* event calls a method named in a closure "uploadController.uploadFiles()". You are going to add this closure and method a little later in this blog post.

```
<div class="form-group btn btn-primary">
  <label for="fileUploadControl">Select Files to Upload</label>
  <input type="file"
         id="fileUploadControl"
         onchange="uploadController.uploadFiles();"
         multiple="multiple"
         style="display:none;" />
</div>
```

Add Area to Display the File Upload Progress

Below the div that contains the file input element, add another div with an *id* attribute set to "fileUploadProgressArea". Display a label and a progress bar for each file the user selects to upload within this div.

```
<!-- BEGIN: File Upload Progress Area -->
<div id="fileUploadProgressArea">
  <!-- This area is where each file to be uploaded is displayed -->
</div>
<!-- END: File Upload Progress Area -->
```

Add Template for each Progress Bar

Below the file upload progress area, add a <script> tag. Set the *id* attribute to "fileUploadProgressTemplate" for this script tag. Also, set the *type* attribute to "text/html". This lets the browser know that this script tag does not contain any executable JavaScript, but plain text that should be ignored. The HTML contained within this script tag is used to display the file name being uploaded and the progress bar as shown in Figure 1.

```
<!-- BEGIN: Display File Upload Progress Template -->
<script id="fileUploadProgressTemplate" type="text/html">
  <div class="row progress-extra-space">
    <div class="col-md-3">
      <label id="fileUploadLabel" class="text-info"></label>
    </div>
    <div class="col-md-9">
      <div class="progress">
        <progress id="fileUploadProgressBar"
                  class="progress-bar progress-bar-success"
                  value="0"
                  max="100"
                  style="width: 100%">
        </progress>
      </div>
    </div>
  </div>
</script>
<!-- END: Display File Upload Progress Template -->
```

You are going to write some JavaScript/jQuery code to clone the HTML within this script tag. Once cloned, you are going to insert the cloned HTML into the div tag you created earlier with the *id* of "fileUploadProgressArea".

Add jQuery and Bootstrap

You are going to need to include references to the jQuery and Bootstrap JS files on your page. Add two more script tags to reference jQuery and Bootstrap from within the Scripts folder where they are installed by NuGet.

```
<script src="/Scripts/jquery-3.4.1.min.js"></script>
<script src="/Scripts/bootstrap.min.js"></script>
```

Make sure you check to see what version of jQuery is loaded after you update all the packages using the NuGet Package Manager. You may need to change the version of jQuery listed above. Also, strictly speaking, Bootstrap does not need to be included on this page for this sample, but I like to include it just in case I use some features of Bootstrap later.

The uploadController Closure

A best practice in JavaScript is to create closures around code. The *uploadController* closure you attached earlier to the onchange event of the file input element is what you are going to write now. To build this closure, add a new <script> tag and add the closure and the stubs of the various methods you are going to write to upload files, display the file name, and show the progress bar.

```
<script>
  var uploadController = (function () {
    /*************************
    * Private Functions
    *************************/
    function uploadFiles() {
    }

    function cloneProgressBarTemplate(fileToUpload, index) {
    }

    function postFile(data, index) {
    }

    function updateProgressBar(e, index) {
    }

    /*************************
    * Public Functions
    *************************/
    return {
      uploadFiles: uploadFiles
    }
  })();
</script>
```

The uploadFiles() method is the only one that is exposed from this closure. The others are called by the uploadFiles() method. Let's build each one of these methods now.

uploadFiles() Method

Once the user selects one or more files using the file input element, the onchange event is fired. That event calls the uploadFiles() method. The first thing to do is to grab the collection of files by accessing the files method from the file input element that has the id "fileUploadControl".

```
function uploadFiles() {
  // Get collection of files selected by user
  var files = $("#fileUploadControl")[0].files;

  // Loop through collection of files selected by user
  for (var index = 0; index < files.length; index ++) {
    // Create new <div> for displaying file name and progress bar
    cloneProgressBarTemplate(files[index], index);

    // Create FormData to post to Web API
    var data = new FormData();
    data.append("fileUploadObject", files[index]);

    // Post the form data
    postFile(data, index);
  }
}
```

The file object, file[index] has properties such as name, size, type, and lastModifiedDate. It is this object that is passed to the Web API method as a file object to be uploaded.

The first thing you do with the file object is pass it to the cloneProgressBarTemplate() method to build a label and a progress bar for the current file object to upload. Next, build a FormData object and append the file object to be uploaded. Set the name of this file object to "fileUploadObject". This name is important when you write your Web API method call as it uses this name to retrieve the file object. Finally, the postFile() method is called to make the Ajax call to the Web API and post the file data to the Web API method.

cloneProgressBarTemplate() Method

As you can see in Figure 1, the file name and a progress bar are displayed for each file selected by the user. The cloneProgressBarTemplate() method is responsible for creating a label and a progress element. Instead of writing JavaScript or jQuery code to create each of the elements required for the label and progress element, you created the complete HTML within the <script> tag with the *id* attribute set to "fileUploadProgressTemplate". It is much easier to layout the HTML within an HTML editor than it is to write the same code using JavaScript.

To create a new DOM element from the HTML within the <script> tag, use the following line of code.

```
var elem = $($("#fileUploadProgressTemplate").html()).clone();
```

The variable *elem* is now a new DOM object and you can append that where ever you wish using the append() method. For example, you created an empty <div> tag earlier in this blog post with the *id* attribute set to "fileUploadProgressArea". Use the following line of code to insert this newly cloned DOM object into that <div> tag.

```
$("#fileUploadProgressArea").append(elem);
```

If you only needed a single DOM object for a single file, this is all the code you would need. However, you can have multiple file objects selected by the user, so you need to build multiple DOM objects. As you know, you are not allowed to have multiple HTML elements with the same *id* attribute value. Therefore, you pass in the index value of each file along with the file object. You use this index number to create a unique *id* attribute value after you have cloned the HTML.

There are two HTML elements within the <script> tag you are cloning that have *id* attributes; "fileUploadLabel" and "fileUploadProgressBar". You are going to use the find() method on the new DOM element to locate these elements. You then set the *id* attribute to the name of the element plus an underscore (_) plus the index number. Thus, the first file passed to this method from the uploadFiles() method passes a zero (0) in the index, so the *id* attributes are "fileUploadLabel_0" and "fileUploadProgressBar_0" respectively. The second time through, the id attributes are "fileUploadLabel_1" and "fileUploadProgressBar_1", and so on.

```
function cloneProgressBarTemplate(fileToUpload, index) {
  // Clone the HTML from the template
  var elem = $($("#fileUploadProgressTemplate").html()).clone();

  // Replace the 'id' attributes with current index number
  $(elem).find("#fileUploadLabel").attr("id",
    "fileUploadLabel_" + index);
  $(elem).find("#fileUploadProgressBar").attr("id",
    "fileUploadProgressBar_" + index);

  // Append this new HTML to the file upload area
  $("#fileUploadProgressArea").append(elem);

  // Display the file name
  $("#fileUploadLabel_" + index).text(fileToUpload.name);
}
```

postFile() Method

After you have displayed the file name in the label, and you have a progress element on the page with unique id's you are ready to post the file to the Web API. You must pass the FormData object to the postFile() method, but it is very important you pass the index number of the current file being processed. This number is what is used to update the appropriate progress element on the page.

You can see in the xhr function of the Ajax call you setup an event listener for the progress of the file upload. When the Web API sends a status of how much of the file has been uploaded, this data is passed as argument *e* in the function of the event listener. Pass this *e* argument and the index to the updateProgressBar() method to update the correct progress element on the web page.

```
function postFile(data, index) {
  // Make the Ajax call
  $.ajax({
    url: "/api/FileUpload/UploadFile",
    type: 'POST',
    data: data,
    contentType: false,
    processData: false,
    xhr: function () {
      var req = $.ajaxSettings.xhr();
      if (req.upload) {
        // Setup event listener for any progress info
        // returned from the Web API
        req.upload.addEventListener('progress', function (e) {
          if (e.lengthComputable) {
            // Update the progress bar
            updateProgressBar(e, index);
          }
        }, false);
      }
      return req;
    },
  }).done(function (response) {
    console.log(response);
  }).fail(function (error) {
    console.log(error);
  });
}
```

updateProgressBar() Method

The updateProgressBar() method calculates the percentage of the data that has been uploaded to the server. It uses this percentage value to set the value of the progress bar element. This forces the browser to update the progress bar. Once the percentage is greater than or equal to 100, set the progress bar value to 100 to ensure it shows the file has been completely uploaded. If you want, you can add another label below the progress bar and once the data is completely uploaded, you can display that label with some text such as "File Upload Complete"

```
function updateProgressBar(e, index) {
  // Calculate current percentage
  var percentage = (e.loaded * 100) / e.total;

  // Modify the progress bar
  $("#fileUploadProgressBar_" + index).val(percentage);

  // Are we done?
  if (percentage >= 100) {
    $("#fileUploadProgressBar_" + index).val(100);
  }
}
```

The Web API Controller

All the client-side code is in place, so you can create your Web API method to upload your file to now. If you are using Microsoft MVC, right mouse-click on the **Controllers** folder and choose **Add** | **Web API Controller Class (v2.1)** from the context-sensitive menu. Set the name to **FileUploadController** and click the OK button. At the top of the file, remove all the using statements, and just make sure you have the following three at the top.

```
using System.IO;
using System.Web;
using System.Web.Http;
```

Remove all the methods within the FileUploadController class and add the following method signature.

```
public class FileUploadController : ApiController
{
  [HttpPost]
  public int UploadFile()
  {
  }
}
```

Get the File Object

To retrieve the fileUploadObject from the FormData object you created in the client-side code, use the Files[] property on the Request object as shown in the code below. This returns to you an object of the type HttpPostedFile. This object is the .NET equivalent of the file object created by the browser.

```
HttpPostedFile fileToUpload =
  HttpContext.Current.Request.Files["fileUploadObject"];
```

Convert Input Stream into a Byte Array

The posted file contents are sent via an input stream, so you need to read that stream and store the data into an array of bytes. The easiest method to get the bytes of data is to copy into a MemoryStream object. Use the code shown below to request the posted file object and copy the data into a byte array.

```
byte[] contents;

HttpPostedFile fileToUpload =
  HttpContext.Current.Request.Files["fileUploadObject"];

if (fileToUpload != null && fileToUpload.ContentLength > 0)
{
  // Get the uploaded file contents
  using (MemoryStream ms = new MemoryStream())
  {
    fileToUpload.InputStream.CopyTo(ms);
    ms.Position = 0;
    contents = ms.ToArray();
  }
}
```

File Object Properties

Just as the file object on the client had properties such as length and file name, so does the HttpPostedFile object. It is useful to gather that data into an object. Instead of creating a real class, for now, just build an anonymous type and store the data from the HttpPostFile object into that anonymous type.

```
// Gather file information into anonymous type
var fileInfo = new
{
  ContentLength = fileToUpload.ContentLength,
  ContentType = fileToUpload.ContentType,
  FilePath = Path.GetDirectoryName(fileToUpload.FileName),
  FileName = Path.GetFileName(fileToUpload.FileName),
  Contents = contents
};
```

Once you have the contents of the file and the file name, store the file on your file server. The complete UploadFile() method is shown in the code below.

```
public class FileUploadController : ApiController
{
  [HttpPost]
  public int UploadFile()
  {
    byte[] contents;

    // Retrieve file to upload
    HttpPostedFile fileToUpload =
      HttpContext.Current.Request.Files["fileUploadObject"];

    if (fileToUpload != null && fileToUpload.ContentLength > 0)
    {
      // Get the uploaded file contents
      using (MemoryStream ms = new MemoryStream())
      {
        fileToUpload.InputStream.CopyTo(ms);
        ms.Position = 0;
        contents = ms.ToArray();
      }

      // Gather file information into anonymous type
      var fileInfo = new
      {
        ContentLength = fileToUpload.ContentLength,
        ContentType = fileToUpload.ContentType,
        FilePath = Path.GetDirectoryName(fileToUpload.FileName),
        FileName = Path.GetFileName(fileToUpload.FileName),
        Contents = contents
      };

      // Write File to Server File System
      var file = HttpContext.Current.Server
                  .MapPath("/UploadedFiles/" + fileInfo.FileName);
      File.WriteAllBytes(file, contents);
    }

    return 0;
  }
}
```

Summary

In this blog post, you learned to upload multiple files using JavaScript, jQuery, and Microsoft ASP.NET Web API. There is not a lot of code required to perform the actual upload but adding a label and a progress bar for each file selected requires you to get a little creative. Using a template of HTML in a <script> tag is a nice way to clone HTML and place it into another section of your HTML page. It is much easier to do this than to write HTML code in your JavaScript.

Source Code

NOTE: You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select “Fairway/PDSA Blog,” then select “Upload Multiple Files Asynchronously - Part 1” from the dropdown list.