Ajax and binary files - downloading files and monitoring progress

We won't go into too much detail here, but recent browsers (> 2012) usually support *XHR2*, that adds the option to directly download binary data. (by setting the responseType property of the Ajax request to `ArrayBuffer`, as we will see).

HTTP is a text based protocol, and when you upload/download images, videos or any binary file, it is text encoded, then decoded on the fly when arriving on a server or in a browser. For a long time, when using Ajax, these binary files had to be decoded "by hand", using JavaScript code.

With XHR2, you can ask the browser to decode the file you send/receive natively. To do this, when you use an `XMLHttpRequest`to send or receive a file, you need to specify the type of file with a value equal to `arrayBuffer`.

Below is a function that loads a sound sample using`XMLHttpRequest` level 2.

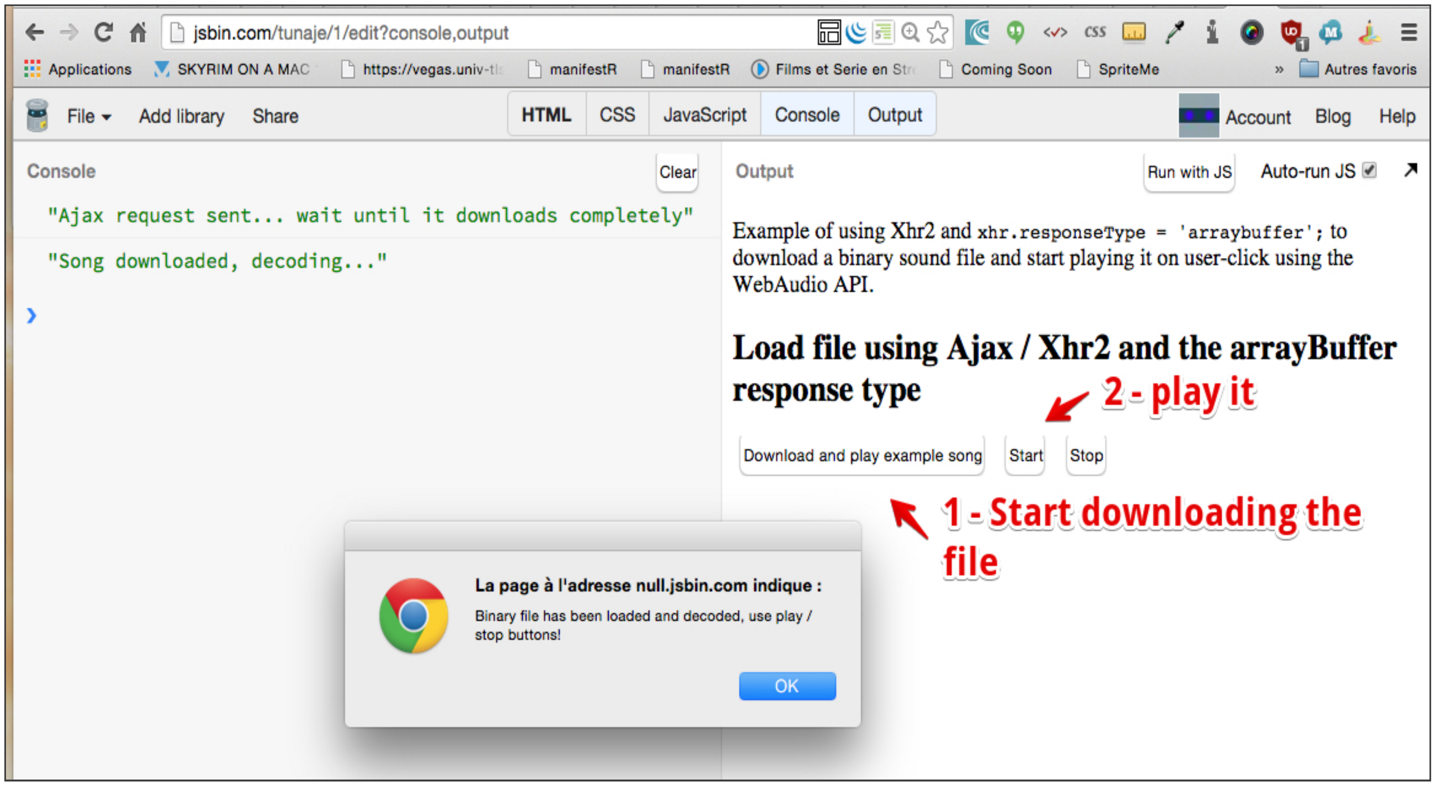
Note: 1) the simple and concise syntax, and 2) the use of the new`arrayBuffer` type for the expected response (*line 5*):

```
    // Load a binary file from a URL as an ArrayBuffer.
    function loadSoundFile(url) {
       var xhr = new XMLHttpRequest();
       xhr.open('GET', url, true);

       xhr.responseType = 'arraybuffer'; // THIS IS NEW WITH HTML5!
       xhr.onload = function(e) {
          initSound(this.response); // this.response is an ArrayBuffer.
       };

 9.    xhr.send();
    }
```

EXAMPLE: DOWNLOAD A BINARY SONG FILE USING XHR2 AND `RESPONSETYPE='ARRAYBUFFER'`, AND PLAY IT USING WEB AUDIO

Try this example on JSBin:

The above function is used, and we modified an example from the HTML5 part 1 course that shows how to read a binary file from disk using the File API's method readAsArrayBuffer. In this example, instead of reading the file from disk, we download it using XHR2.



```
"Ajax request sent... wait until it downloads completely"
"Song downloaded, decoding..."
```

Example of using Xhr2 and `xhr.responseType = 'arraybuffer';` to download a binary sound file and start playing it on user-click using the WebAudio API.

Load file using Ajax / Xhr2 and the arrayBuffer response type

La page à l'adresse null.jsbin.com indique :

Binary file has been loaded and decoded, use play / stop buttons!

Complete source code:

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <title>XHR2 and binary files + Web Audio API</title>
 </head>
<body>
 <p>Example of using XHR2 and<code>xhr.responseType = 'arraybuffer';
</code> to download a binary sound file
and start playing it on user-click using the Web Audio API.</p>
 <p>
 <h2>Load file using Ajax/XHR2 and the arrayBuffer response
type</h2>
 <buttononclick="downloadSoundFile('http://myserver.com/song.mp3');">
     Download and play example song.
 </button>
 <button onclick="playSound()"disabled>Start</button>
 <button onclick="stopSound()"disabled>Stop</button>
<script>
  // WebAudio context
  var context = new window.AudioContext();
  var source = null;
  var audioBuffer = null;

  function stopSound() {
    if (source) {
       source.stop();
    }
  }

  function playSound() {
    // Build a source node for the audio graph
    source = context.createBufferSource();
    source.buffer = audioBuffer;
    source.loop = false;
    // connect to the speakers
    source.connect(context.destination);
    source.start(0); // Play immediately.
  }

  function initSound(audioFile) {
    // The audio file may be an mp3 - we must decode it before
playing it from memory
    context.decodeAudioData(audioFile,function(buffer) {
      console.log("Song decoded!");
      // audioBuffer the decoded audio file we're going to work with
      audioBuffer = buffer;
      // Enable all buttons once the audio file is
      // decoded
      var buttons =document.querySelectorAll('button');
      buttons[1].disabled = false; // play
      buttons[2].disabled = false; // stop
      alert("Binary file has been loaded and decoded, use play /
stop buttons!")
    }, function(e) {
       console.log('Error decoding file', e);
    });
  }

  // Load a binary file from a URL as an ArrayBuffer.
```

```
    function downloadSoundFile(url) {
      var xhr = new XMLHttpRequest();
      xhr.open('GET', url, true);

62.   xhr.responseType = 'arraybuffer'; // THIS IS NEW WITH HTML5!
      xhr.onload = function(e) {
         console.log("Song downloaded, decoding...");
         initSound(this.response); // this.response is an ArrayBuffer.
      };
      xhr.onerror = function(e) {
        console.log("error downloading file");
      }

      xhr.send();
72.      console.log("Ajax request sent... wait until it downloads
completely");
    }
</script>
</body>
</html>
```

Explanations:

- *Line 12*: a click on this button will call the `downloadSoundFile`function, passing it the URL of a mp3 file.
- *Lines 58-73*: this function sends the Ajax request, and when the file has arrived, the `xhr.onload` callback is called (*line 63*).
- The initSound function decodes the mp3 into memory using the WebAudio API, and enables the play and stop buttons.
- When the play button is enabled and clicked (*line 15*) it calls the`playSound` function that builds a minimal Web Audio graph with a `BufferSource` node that contains the decoded sound (*lines 31-32*), and connects it to the speakers (*line 35*) then plays it.

MONITORING UPLOADS OR DOWNLOADS USING A`PROGRESS` EVENT

1 - Declare a `progress` event handler

XHR2 now provides `progress` event attributes for monitoring data transfers. Previous implementations of `XmlHttpRequest` didn't tell us anything about how much data has been sent or received. TheProgressEvent interface adds 7 events that we can listen to while downloading or uploading files.

attribute	type	Explanation
`onloadstart`	`loadstart`	When the request starts.
`onprogress`	**`progress`**	**While loading and sending data.**
`onabort`	`abort`	When the request has been aborted, either by invoking the `abort()` method or navigating away from the page.
`onerror`	`error`	When the request has failed.
`onload`	`load`	When the request has successfully completed.
`ontimeout`	`timeout`	When the author specified timeout has passed before the request could complete.
`onloadend`	`loadend`	When the request has completed, regardless of whether or not it was successful.

The syntax for declaring progress event handlers is slightly different depending on the type of operation: a download (using the GET HTTP method), or an upload (using POST).

Syntax for download:

```
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
...
xhr.onprogress = function(e) {
 // do something
}

xhr.send();
```

Note that an alternative syntax such as`xhr.addEventListener('progress', callback, false)` also works.

Syntax for upload:

```
var xhr = new XMLHttpRequest();
xhr.open('POST', url, true);
...
xhr.upload.onprogress = function(e) {
 // do something
}

xhr.send();
```

Notice that the only difference is the "upload" added after the name of the request object: with GET we use `xhr.onprogress` and with POST we use `xhr.upload.onprogress`.

Note that an alternative syntax such as`xhr.upload.addEventListener('progress', callback, false)` also works.

2 - Get progress values (how many bytes have been downloaded) and the total file size

The event `e` passed to the `onprogress` callback has two interesting properties:

1. `loaded` that corresponds to the number of bytes that have been downloaded or uploaded so far by the browser, and
2. `total` that corresponds to the total number of bytes of the file.

Using it altogether with a `<progress>` element, it is very easy to render an animated progression bar. Here is a source code extract that does this for a download (full working example later in this page):

HTML:

```
<progress id="downloadProgress" value=0><progress>
```

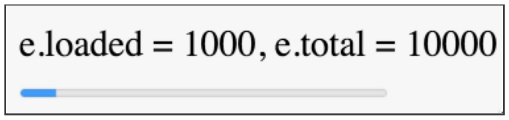
JavaScript:

```
// progress element
var progress =document.querySelector('#downloadProgress');
```

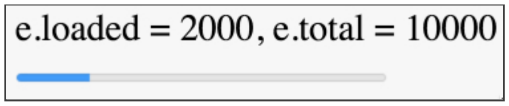
```
function downloadSoundFile(url) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', url, true);

  ...
  xhr.onprogress = function(e) {
    progress.value = e.loaded;
    progress.max = e.total;
  }
  xhr.send();
}
```

Explanations: by setting the `value` and `max` attributes of the`<progress>` element with the current number of bytes downloaded by the browser and with the total size of the file (*lines 10-11*), it makes the `<progress>` element reflect the actual percentage of progression.

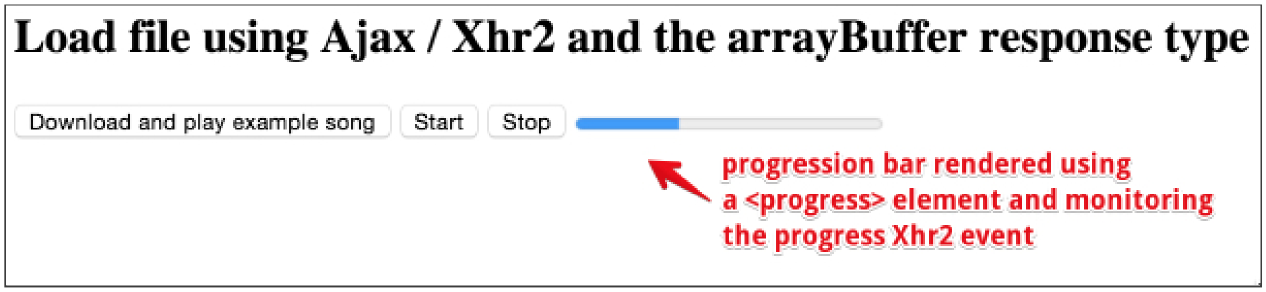
For example, with a file that is 10,000 bytes long, if the current number of bytes downloaded is 1000, then `<progress value=1000 max=10000>` will look like this:



And a current download of 5000 bytes will give `<progress value=5000 max=10000>` and will look like this:



COMPLETE EXAMPLE: MONITORING THE DOWNLOAD OF A SONG FILE


Load file using Ajax / Xhr2 and the arrayBuffer response type
Download and play example song
Start
Stop
progression bar rendered using a <progress> element and monitoring the progress Xhr2 event

This is a variant of the previous example that uses the `progress`event and a `<progress>` HTML5 element to display an animated progression bar while the download is going on.

Try it on JSBin - look at the code, which includes the previous source code extract.