

Read file content as binary

INTRODUCTION

This method is rarely used, except for loading "raw" binary data. For images you would like to see in your HTML page using the `` or for drawing in a canvas, or for audio and video files that you would like to play using the `<audio>` or `<video>` elements, it would be preferable to use the `readAsDataURL` method presented on the next page of the course.

`readAsArrayBuffer` is often used for purposes such as reading audio samples that should be loaded in memory and played using the WebAudio API, or for loading textures that you will use with WebGL for 3D animations.

EXAMPLE 1: READ A LOCAL AUDIO FILE AND PLAY IT WITH THE WEBAUDIO API

The WebAudio API is useful for reading audio sound samples from memory (no streaming), and has been designed for music application and games. This example shows how a local audio file can be read and played directly in the browser, without the need for a server!

[Example on JS Bin](#) (does not work on IE, as it does not support the WebAudio API). We could not embed it here on the edX platform as it prevents code that uses Ajax to run in its pages.

The screenshot shows a JSBin editor interface. The code in the main editor is as follows:

```
function initSound(audioFile) {  
  // The audio file may be a mp3, we must decode it before  
  playing it from memory  
  context.decodeAudioData(audioFile, function(buffer) {  
    // audioBuffer the decoded audio file we're going to work  
    with  
    audioBuffer = buffer;  
  
    // Enable all buttons once it's decoded  
    var buttons = document.getElementsByTagName('button');  
    buttons[0].disabled = false;  
    buttons[1].disabled = false;  
  }, function(e) {  
    console.log('Error decoding file', e);  
  });  
}  
  
// User selects file, read it as an ArrayBuffer and pass to the  
// API.  
var fileInput = document.querySelector('input[type="file"]');  
fileInput.addEventListener('change', function(e) {  
  var reader = new FileReader();  
  
  reader.onload = function(e) {  
    initSound(e.target.result);  
  };  
  // THIS IS THE INTERESTING PART!  
  reader.readAsArrayBuffer(this.files[0]);  
}, false);  
</script>  
</body>  
</html>
```

Annotations on the image:

- 1 - choose an audio file, it will be read as binary**: Points to the `reader.readAsArrayBuffer(this.files[0]);` line.
- 2 - pass its content to initSound**: Points to the `initSound(e.target.result);` line.
- 3 - decode it and play it in memory using WebAudio**: Points to the `context.decodeAudioData(audioFile, ...)` line.

The right sidebar shows the 'Output' panel with the text: "Example of using the Web Audio API to load a sound file and start playing on user-click." and "The WebAudio API is beyond the scope of this course (covered in the HTML5 part 2 course), but notice that the local file is read as a binary file with `reader.readAsArrayBuffer`."

Source code extract:

```
// User selects file. Read it as an ArrayBuffer and pass to  
the API.  
var fileInput = document.querySelector('input[type="file"]');  
fileInput.addEventListener('change', function(e) {  
  var reader = new FileReader();
```

```
    reader.onload = function(e) {  
        initSound(e.target.result);  
    };  
10. // THIS IS THE INTERESTING PART!  
    reader.readAsArrayBuffer(this.files[0]);  
    }, false);
```

Explanations:

- *Line 1:* we get a pointer to the file selector, the variable `fileInput`.
- *Line 4:* we define a `change` listener. In this example, we use an anonymous function directly included in the listener definition (the listener is the `function(e) { ... }`).
- *Line 11:* when a user chooses a file, the listener will be executed. Line 11 will start the reading of the file content, as a binary file (this is what `readAsArrayBuffer` means: read as binary!). Once the file will be entirely read, the `onload` callback will be asynchronously called by the browser.
- *Line 7* is the `onload` callback, executed when the file content is loaded in memory. We pass the file content to the `initSound` function (see JS Bin example for complete source code) that uses `WebAudio` to decode it (it may be a compressed file - an mp3 for example - and `WebAudio` works only with uncompressed audio formats in memory), and to play it.