[**What is JSON?**](https://courses.edx.org/courses/course-v1:W3Cx+JS.0x+3T2017/courseware/b108897ce3c4449c8326cda572490b8f/ed3ab9994af649e6a1e40ba6d80a8ff5/?child=first)

JSON stands for**J**ava**S**cript **O**bject **N**otation. It's a standard for transforming nearly any object into a string representation that is human readable. It became a standard for exchanging data to/from a remote HTTP server, and is available for many other languages in addition to JavaScript.

A JavaScript object o in JSON looks a lot like what o.toString() returns.

**There are two main methods to know:**

* 1. Transform any JavaScript object in JSON:                      **var jsonStr = JSON.stringify(obj);**
  2. Transform any JSON string into a JavaScript object:    **var jsObj  = JSON.parse(jsonStr);**

**Let's see some examples:**

1. > var x = 3;
2. undefined
4. >**JSON.stringify(x);**
5. "3"
7. > var simpleObject = {x:12, y:30};
8. undefined
10. >**JSON.stringify(simpleObject);**
11. "{"x":12,"y":30}"
13. > var anArray = ['Monday', 'Tuesday', 'Wednesday'];
14. undefined
16. >**JSON.stringify(anArray);**
17. "["Monday","Tuesday","Wednesday"]"
19. > var complexObject = {name:'Metallica',
20. albums:[
21. {name:"Master of Puppets", year:1986},
22. {name:"Black Album", year:1991}
23. ]
24. };
25. undefined
27. >**JSON.stringify(complexObject);**
28. "{"name":"Metallica","albums":[{"name":"Master of Puppets","year":1986},{"name":"BlackAlbum","year":1991}]}"

In the above examples, you can see JSON representations of a simple variable of a predefined type, of an array, of a simple object, of an object that contains an array of objects (Metallica example).

And indeed, it looks like the code you typed to create the objects, with quotes around it and around the property names. This is why it is called JavaScript Object Notation ;-)

**You cannot use JSON objects as JavaScript objects**

The JSON representation of JavaScript objects is a string. JSON has been developed mainly for replacing XML as a format for exchanging data between a client and a remote HTTP server. It has become very popular as the format for exchanging data when a Web Application uses Ajax for its communications with the HTTP server. **Ajax is a way to send / receive data in the background, without the need to reload the Web page. Along with the DOM API, the appearance of this technology in 2005 with Internet Explorer, made it possible to make Web pages more dynamic. Google Maps was one of the first popular Ajax-powered Web application: as you moved the map, new parts arrived (downloaded in the background from the Gmap HTTP server), and the page updated to display these new parts**.

**JSON is also very practical for storing objects where strings are expected. There is a data store in your browser called LocalStorage that can be used as a small database for Web applications, but it stores only pairs of key/values in the string format. If you want to save a JavaScript object, you will have to turn it into JSON, then save it. When you read it from the data store, you will need to turn it back from JSON to JavaScript.**

**Here is a first example that turns an object into JSON and back into a JavaScript object**:

1. > var metallica = {name:'Metallica', albums:[{name:"Master of Puppets", year:1986},   
          {name:"Black Album", year:1991}]};
2. undefined
3. > var metallicaJSON = JSON.stringify(metallica);
4. undefined
5. > metallicaJSON;
6. "{"name":"Metallica","albums":[{"name":"Master of Puppets","year":1986},  
          {"name":"Black Album","year":1991}]}"
7. > metallicaJSON.name; // metallicaJSON is not a JavaScript object
8. undefined
9. > metallica.name; // metallica is an object
10. "Metallica"
11. > var obj = JSON.parse(metallicaJSON); // JSON -> object
12. undefined
13. > obj.name; // this is an object
14. "Metallica"

**With the JSON representation of an object you cannot access the original object's properties using the "." operator, nor call its methods. The JSON format only stores the list of the object properties (name and value) as a string. Look at *line 10*: we cannot access the name property of the JSON representation of the metallica object defined at *line 1*.**

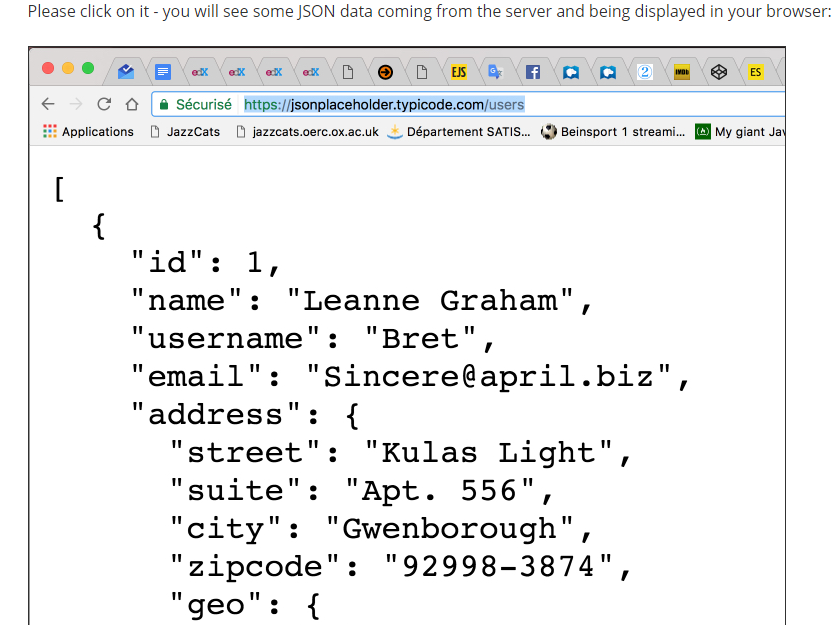
**When we parse a JSON string using JSON.parse(), we get a real JavaScript object, and we can access its properties (*lines 16 and 19*).**

### Example: consuming remote data

#### JSON data from a REST WebService

Most "big sites" provide what we call a REST API. This means "they propose to send/receive data to/from programs over HTTP", and most of the time the JSON format is one of the possible transport formats for the data. Google APIs, Facebook and Amazon APIs are like this.

JSONPlaceholder is a free online REST service that you can use whenever you need some fake data in JSON. Faking a server is great for tutorials, and this is exactly what the next example does. It will consume data from [this URL](https://jsonplaceholder.typicode.com/users).

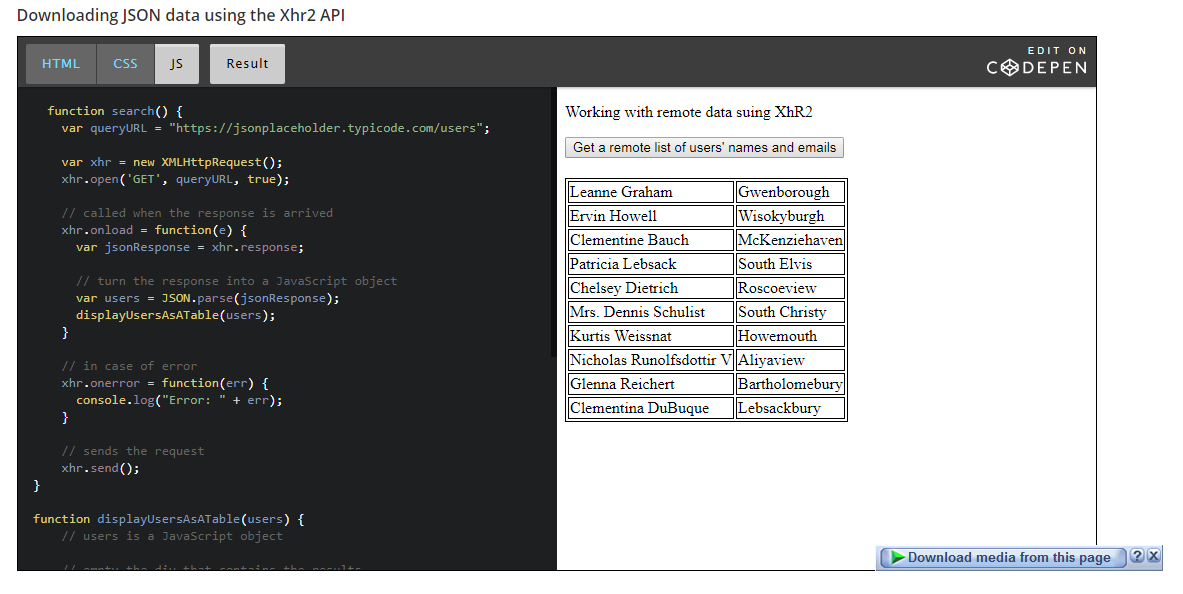


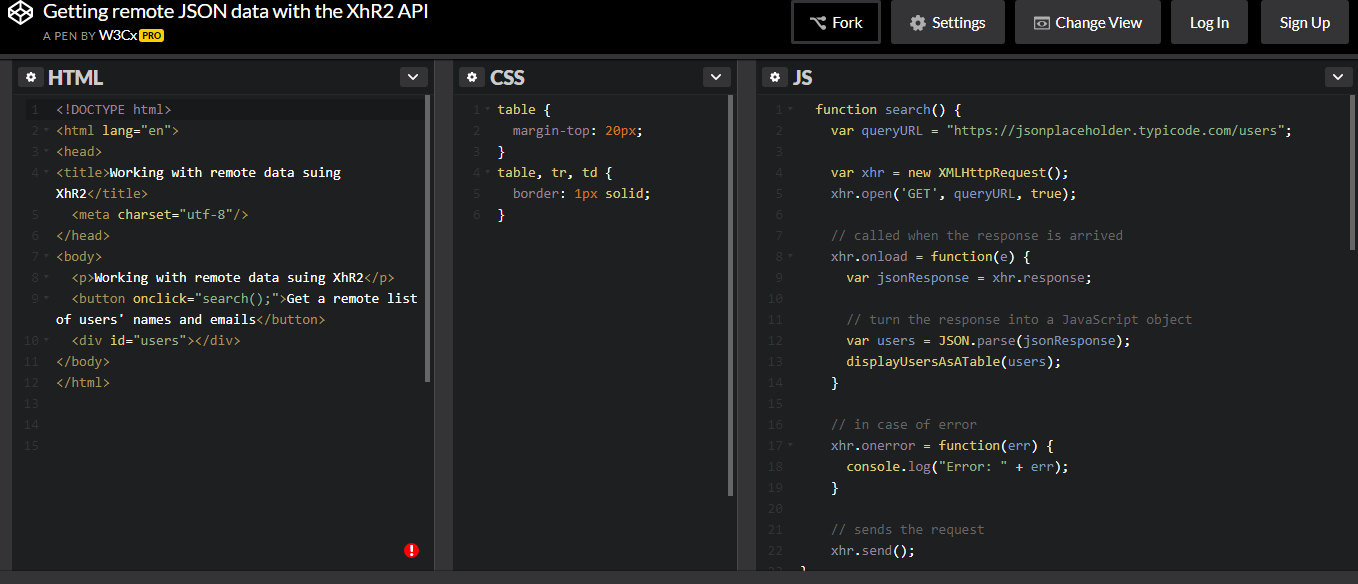
And we would like to use these data in our code, manipulating them as a JavaScript object.

#### First thing: get the remote data as JSON

This course will not cover Ajax and what we call "asynchronous JavaScript". This will be covered in an advanced course to come on W3Cx.

However, we can show you two simple examples that use the Xhr2 API for Ajax requests and the new fetch API that is simplest to use.





Javascript code –

function search() {

var queryURL = "https://jsonplaceholder.typicode.com/users";

var xhr = new XMLHttpRequest();

xhr.open('GET', queryURL, true);

// called when the response is arrived

xhr.onload = function(e) {

var jsonResponse = xhr.response;

// turn the response into a JavaScript object

var users = JSON.parse(jsonResponse);

displayUsersAsATable(users);

}

// in case of error

xhr.onerror = function(err) {

console.log("Error: " + err);

}

// sends the request

xhr.send();

}

function displayUsersAsATable(users) {

// users is a JavaScript object

// empty the div that contains the results

var usersDiv = document.querySelector("#users");

usersDiv.innerHTML = "";

// creates and populate the table with users

var table = document.createElement("table");

// iterate on the array of users

users.forEach(function(currentUser) {

// creates a row

var row = table.insertRow();

// insert cells in the row

var nameCell = row.insertCell();

nameCell.innerHTML = currentUser.name;

var cityCell = row.insertCell();

cityCell.innerHTML = currentUser.address.city;

});

// adds the table to the div

usersDiv.appendChild(table);

}

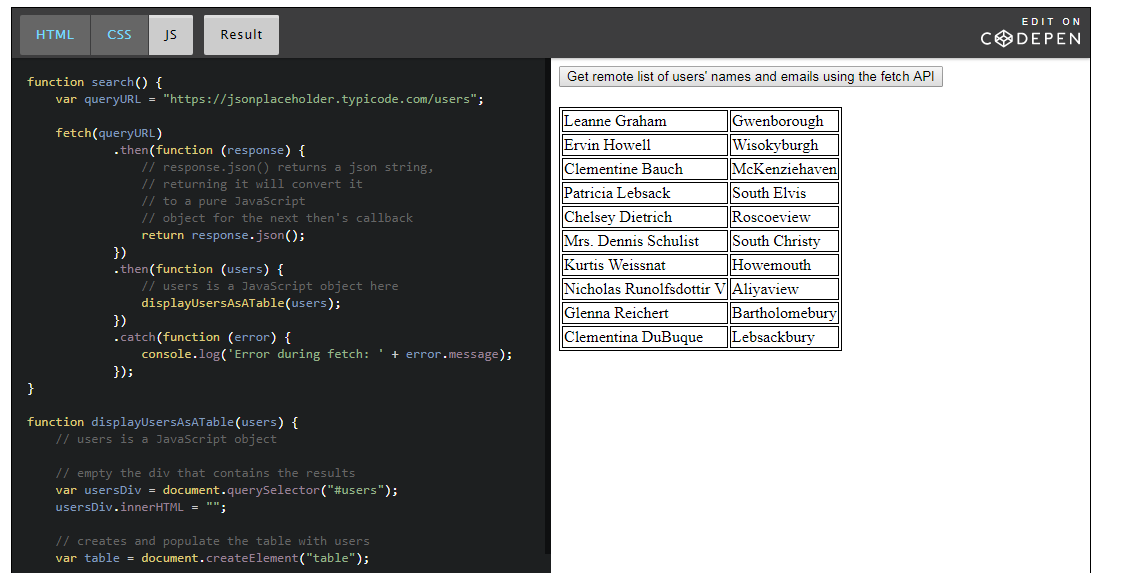
**Explanations**:

* + *Lines 4 and 5* build an Ajax request using XhR2.
  + *Line 22* is executed after: the request is sent in the background (we say "asynchronously").
  + *Line 8*: when the server answers, this callback is executed, and inside it, this.response corresponds to the response from the HTTP server. It's in JSON format (*line 9*)
  + *Line 12*: we turn the JSON response into a regular JavaScript object we can work with, using JSON.parse().
  + *Line 13*: we pass this list of users, now a JavaScript object, to the displayUsersAsATable method, that will use the HTML table API we saw earlier in the course.

#### [advanced] Downloading JSON data using the fetch API

The [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/GlobalFetch/fetch) method that provides an easy, logical way to fetch resources asynchronously across the network. You **fetch** data from a URL, **then**, you do something with the response, **then** you do something else. If there is an error you can **catch** this error and display, for [example](https://courses.edx.org/courses/course-v1:W3Cx+JS.0x+3T2017/courseware/b108897ce3c4449c8326cda572490b8f/ed3ab9994af649e6a1e40ba6d80a8ff5/?child=first), an error message.

See [this blog post](https://davidwalsh.name/fetch) for a detailed tutorial. Asynchronous JavaScript and JavaScript promises (the fetch...then...then... is based on the concept of "promises"), will be detailed in a next MOOC to appear at W3Cx.



JavaScript source code extract:

1. function search() {
2. var queryURL = "https://jsonplaceholder.typicode.com/users";
3. fetch(queryURL)
4. .then(function(response) {
5. **// response is a json string,**
6. **// convert it to a pure JavaScript object**
7. **return response.json();**
8. })
9. .then(function(users) {
10. **// users is a JavaScript object here**
11. displayUsersAsATable(users)
12. })
13. .catch(function(error) {
14. console.log('Error during fetch: ' + error.message);
15. });
16. }

The fetch API will also be covered in an advanced JavaScript course to come. In contrast to XhR2, fetch is based on a concept called "JavaScript promises" (also covered in the advanced course!). You recognize promises when you see ".then..." ".then...".

### Example of use: the LocalStorage API as a client-side database for JavaScript objects

#### The Web Storage API (localStorage, sessionStorage)

The Web storage API (see the [related W3C specification](http://www.w3.org/TR/webstorage/)) introduces "two related mechanisms, similar to HTTP session cookies, for storing structured data on the client side".

Indeed, Web Storage provides two interfaces - sessionStorage and localStorage - whose main difference is data longevity. This specification defines an API for persistent data storage of key-value pair data in Web clients.

**With localStorage the data will remain until it is deleted, whereas with sessionStorage the data is erased when the tab/browser is closed.**

For convenience, we will mainly illustrate the localStorage object. Just change "local" to "session" and it should work (this time with a session lifetime).

#### Simple key-value stores, one per domain (following the [same origin policy](https://en.wikipedia.org/wiki/Same-origin_policy))!

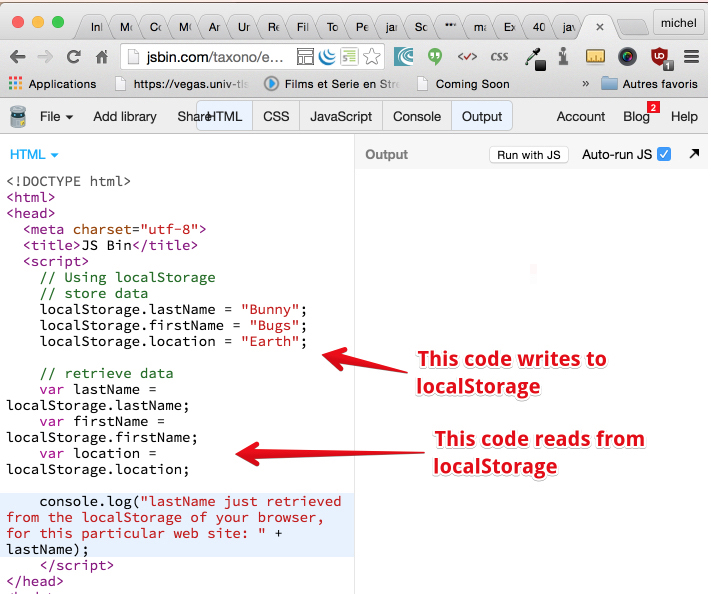
localStorage is a simple key-value store, in which the keys and values are strings. There is only one store per domain. This functionality is exposed through the globally available localStorage object. The same applies to sessionStorage.

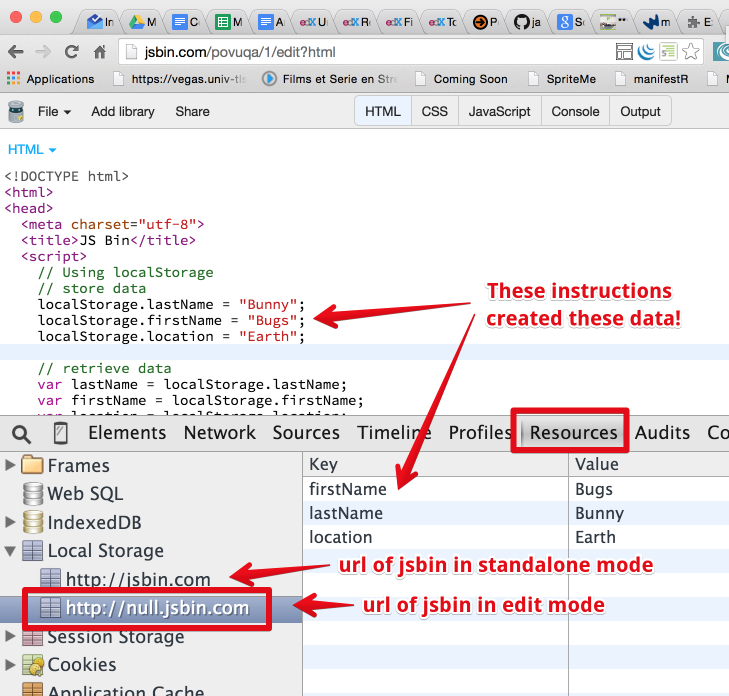
Example:

1. // Using localStorage
2. // store data
3. localStorage.lastName = "Bunny";
4. localStorage.firstName = "Bugs";
5. localStorage.location = "Earth";
6. // retrieve data
7. var lastName = localStorage.lastName;
8. var firstName = localStorage.firstName;
9. var location = localStorage.location;

This data is located in a store attached to the origin of the page. We've created [a JsBin example in which we've included the above code](http://jsbin.com/hebino/1/edit?html,output).

Once opened in your browser, the JavaScript code is executed. With the browser dev. tools, we can check what has been stored in the localStorage for this domain:





#### Differences with cookies?

Cookies are also a popular way to store key-value pairs. Web Storage, however, is a more powerful technique than cookies. The main difference is in size limits: cookies are limited to a few KBytes whereas Web Storage may extend to several MBytes. Also, cookies generate additional HTTP request traffic (whether to request a Web page, an image, a stylesheet, a JavaScript file, etc.).

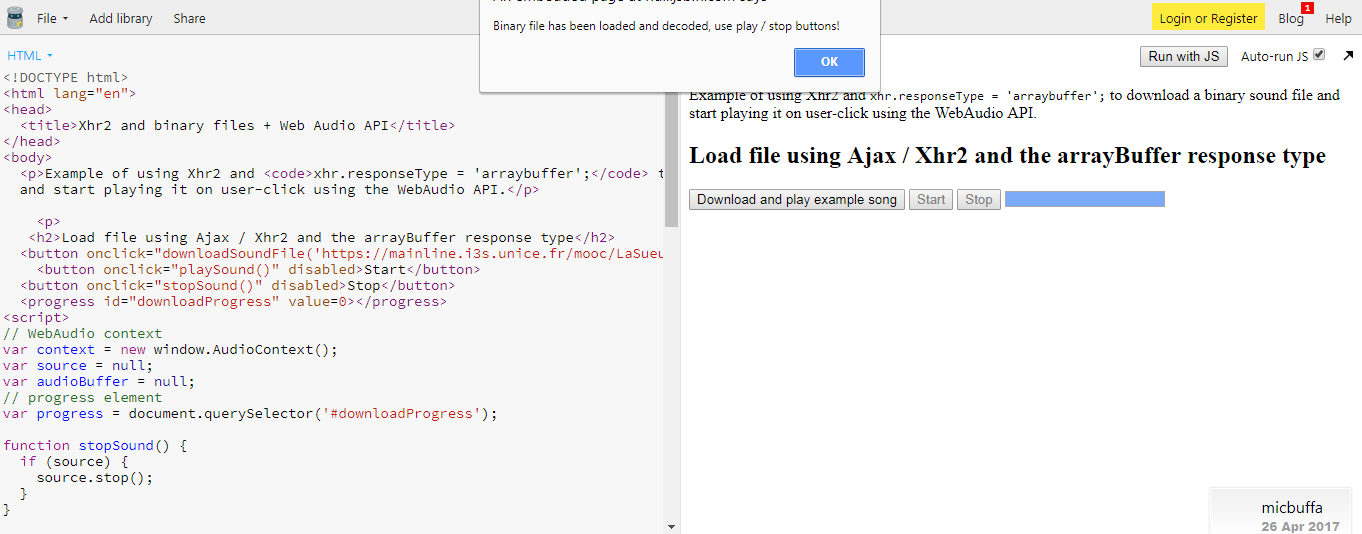
Objects managed by Web Storage are no longer carried on the network and HTTP, and are easily accessible (read, change and delete) from JavaScript, using the Web Storage API.

#### External resources

* [The W3C Web Storage API recommendation (published on 9 June 2015)](http://www.w3.org/TR/webstorage/)
* [Interesting article on html5rocks that compares the different ways of doing client side persistence with HTML5, including Web Storage.](http://www.html5rocks.com/en/tutorials/offline/storage/)

**Important –**

* Did you know that using XhR2 for getting remote data, you can also monitor the download progression using n xhr.onprogress = event listener and a <progress> HTML element. See [this example from the HTML5 Apps and Games MOOC](https://jsbin.com/nuxanaf/edit?html,output)! This is not yet possible using [the new fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

****

**HTML –**

**<!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 WebAudio API.</p>**

**<p>**

**<h2>Load file using Ajax / Xhr2 and the arrayBuffer response type</h2>**

**<button onclick="downloadSoundFile('https://mainline.i3s.unice.fr/mooc/LaSueur.mp3');">Download and play example song</button>**

**<button onclick="playSound()" disabled>Start</button>**

**<button onclick="stopSound()" disabled>Stop</button>**

**<progress id="downloadProgress" value=0></progress>**

**<script>**

**// WebAudio context**

**var context = new window.AudioContext();**

**var source = null;**

**var audioBuffer = null;**

**// progress element**

**var progress = document.querySelector('#downloadProgress');**

**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 a 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);**

**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.onprogress = function(e) {**

**progress.value = e.loaded;**

**progress.max = e.total;**

**}**

**xhr.onerror = function(e) {**

**console.log("error downloading file");**

**}**

**xhr.send();**

**console.log("Ajax request sent... wait until it downloads completely");**

**}**

**</script>**

**</body>**

**</html>**