Setup and Context

1. Setup Context
   1. The Server Site aka NotDelicous.
      1. <http://lightsout.co>
      2. Show the Site
      3. Why are we going to use it?
      4. show the JSON EndPoint
         1. <http://lightsout.co/services/getlinks>
   2. What is DeveloperSmackdown?
      1. Show the Site
      2. Why are we going to use it?
      3. Show the oData services @ <http://developerSmackdown.com/services/odata>

Simple XHR Call - 1-xhr.cshtml

1. Open 1-xhr.cshtml
   1. Purpose:
      1. Make a simple xhr2 call to another resource.
2. Show the response header from
   * 1. <http://lightsout.co> and \or <http://developersmackdown.com>

Access-Control-Allow-Origin: \*

1. Now make a simple call against the

var xhr = new XMLHttpRequest();

xhr.open('GET', 'http://developersmackdown.com');

xhr.onload = callback;

xhr.send();

1. Our callback function will take care of the response ( xhr.onload )

var callback = function(e) {

$('#results').append(this.responseText);

}

1. Simple enough but all we really did was a get on some html.

JSON CORS call with XHR - 2-xhr.cshtml

1. Open 2-xhr
   1. Now we’re going to make a true services call to <http://DeveloperSmackdown.com> and we are going to tell the service we expect JSON returned via a header.
2. First let’s make the call in Fiddler to see how the service works.
   1. Open fiddler.
      1. Make the call without the header. This will return xml by default.

<http://developersmackdown.com/services/odata/Shows(PodcastId=1,ShowId=54)>

* + 1. Make the same call with the accept header added.

accept: application/json

1. Create the XHR request

var xhr = new XMLHttpRequest();

xhr.open('GET', 'http://developersmackdown.com/services/odata/Shows(PodcastId=1,ShowId=54)');

xhr.setRequestHeader('accept', 'application/json');

xhr.onload = callback;

xhr.send();

1. Create the callback for onload

var callback = function(e) {

var data = JSON.parse(this.responseText);

$('#results').text(data.d.Title);

}

1. Look in the debugger at the xhr call.

Blob CORS call with XHR - 3-xhr.cshtml

1. Open 3-xhr
   1. One of the great new features of XHR2 is the ability to get easy access to other resources like images.
   2. We’re just going to grab an image and display it.
      1. With XHR2, you full access to the stream to do things like manipulation or even save it to the local file system with the new HTML5 file access api.
2. For this example let’s grab my Gravatar. <http://gravatar.com/clarksell>

window.URL = window.URL || window.webkitURL;

var xhr = new XMLHttpRequest();

xhr.open('GET', 'http://0.gravatar.com/avatar/592fd4bb2692c7d9fbe8f5ef3af52309?size=420', true);

xhr.responseType = 'blob';

xhr.onload = callback;

xhr.send();

1. Now we need the callback to process the response.

var callback = function(e) {

if (this.status == 200) {

var blob = this.response;

var img = document.createElement('img');

img.onload = function(e) {

window.URL.revokeObjectURL(img.src);

};

img.src = window.URL.createObjectURL(blob);

$('#images').append(img);

};

};

JSON.parse 4-json.cshtml

1. JSON.parse
   1. You might have caught that I was using the JSON api earlier when we processed the XHR request. Now let’s look again in detail.
2. 3 methods
   1. JSON.parse – create an object from json
   2. JSON.stringify – create a JSON string representation of an object
   3. .toJSON – augment the JSON serialization process
3. Open /scripts/sampleJSON.js
   1. Just want to show the sample object and sample JSON string
4. Let’s parse a string ( sampleJsonString ) into an object.

var results = JSON.parse(sampleJsonString);

$('#results').text(results.firstName);

1. Easy enough. Now let’s create a convert function that will be called during serialization, and pass that in.

var convert = function ( key, value ) {

if ( key === 'firstName' ) {

value = "something"

}

return value;

};

var results = JSON.parse(sampleJsonString, **convert**);

JSON.strinify 5-json.cshtml

1. JSON.stringify
   1. Taking an object and creating the JSON string representation is rather easy.
   2. We’re going to take the sampleJsonObj and stringify it.

var results = JSON.stringify(sampleJsonObj);

$('#results').text(results);

1. Serialization whitelist
   1. A great feature in the serialization process is something called whitelisting
   2. This is the ability to pass an array of strings, and only those found will be included in the serialization.

var whiteList = ['firstName'];

var results = JSON.stringify(sampleJsonObj, **whiteList**);

1. Then pass in the whitelist array to stringify

obj.toJSON 6-json.cshtml

1. Obj.toJSON
   1. Any object that has toJSON on it will end up getting called during the serialization process.

sampleJsonObj.toJSON = function () {

alert('in custom serializer');

};

Polyfilling JSON Serialization 7-json.cshtml

1. Polyfilling
   1. As with a lot of things in the grander “HTML5” not everything is of course supported. We have a number of browsers we need to support as well as their versions. And since the spec isn’t just one spec but the columniation of a number of thing not every browser has everything implemented.
   2. One practice in dealing with this is ‘polyfilling’. This is typically a practice you see implemented when doing feature detection in the browser for items such as canvas but can also be used for JavaScript.
   3. To do so, we’re going to use a tool called yepnope. We are basically going to test for the JSON object and if it exists then just run our code, if not then we will load the polyfill and then run our code.
2. Open 7-json.cshtml in IE9.
   1. This will run and you will see the string on the page.
   2. Then open the IE Dev Tools ( f12 ) and run in IE7 or IE5 from IE10. Now we will get a script error.
   3. Add the yepnope polyfill and rerun.
      1. Show the polyfill being loaded into the script stack dynamically.
      2. Since the polyfill API is exactly the same as the actual implementation no code needs to be changed.

yepnope({

test: window.JSON,

nope: "/scripts/json2/json2.js",

complete: stringMe

});

Putting it all together in a form. 8-form.cshtml

1. Just to be a bit crazy, let’s put it all together.
   1. We have a form on the localhost site.
      1. This form will create a link on lightout.co
   2. This same page has a link to get a number of links from Developer Smackdown
   3. Once it has those links we post that back to lightsout.co
   4. Then we grab all the links from lightsout.co and display them on our page.
   5. 3 different sites. All CORS enabled.

Strict Mode 9-strict.cshtml

The purpose of strict mode is to prepare you for the future. It’s to enable “the good parts” while preparing you for the next version of ECMAScript.

1. To enable it, just add the string.

"use strict";

1. Since it’s just a string, if the browser doesn’t support it, it will just be ignored.
2. It also has to be the first statement in scope.

"use strict";

foo = "foo";

1. While doing that will cause an error, if you were to flip it around it wouldn’t.
2. “use strict” is based on scope. So here we have the same statements but will need to apply them correctly such that “our code” is using strict and not forcing it upon someone else.

var something = "something else";

"use strict"; // wasn't first statement

foo = "foo"; // will not fail

(function() {

"use strict";

bar = "bar"; // will fail

})();

1. Strict Mode is not enforced on non-strict functions that are invoked inside the body of a strict function

var a = function ( newFunc ) {

'use strict';

newFunc();

};

var b = function () {

undefined = 'asdf';

};

a(b);

1. Inner functions inside the body of a strict function will carry strict mode

var a = function ( ) {

'use strict';

var b = function () {

undefined = 'asdf';

};

return b();

};

a();

1. Easy enough and we should all start using it today. But there can be a catch. A common practice tools like file concatenation and loaders etc.
   1. Let’s explore a few simple files:
      1. /strictScripts/**badScript.js**
      2. /strictScripts/**someScript.js**
      3. /strictScripts/**new.js - empty**
   2. Now let’s use some file concatenation on them, in particular Sprockets.
      1. Once the Sprockets runs, it will actually change the scoping of our files and inadvertently disable strict mode. This isn’t the fault of tool, but rather our poorly written JavaScript in the first place.

sprocketize strictScripts/\*.js >> strictScripts/new.js