
Pocket Guides
Go Make Things
AJAX/HTTP
with Vanilla JavaScript
CHRIS FERDINANDI

Ajax/HTTP

By Chris Ferdinandi

Go Make Things, LLC

v2.0.0

Table of Contents

Intro

In this guide, you'll learn:

- How to make HTTP requests.
- How to work around cross-domain scripting restrictions with JSONP.
- How to use a lightweight micro-library to make working with Ajax and HTTP easier.
- How to get and manipulate elements from another page and inject them into your site.
- How to put it all together and write a working project with Ajax.

Let's get started!

Ajax/HTTP Requests

Ajax requests have really good browser support, but require a few steps and can be a bit tedious to write. In a later chapter, we'll look at a lightweight micro-library that makes working with Ajax a lot easier. But first, let's look at how to make Ajax requests without any help.

It's a three step process:

1. Setup our request by creating a new `XMLHttpRequest()`.
2. Create an `onreadystatechange` callback to run when the request completes.
3. Open and send our request.

For testing purposes, we'll pull data from JSON Placeholder[1], a site that provides real API endpoints with placeholder content.

```
// Set up our HTTP request
var xhr = new XMLHttpRequest();

// Setup our listener to process compeleted requests
xhr.onreadystatechange = function () {
    // Only run if the request is complete
    if ( xhr.readyState !== 4 ) return;

    // Process our return data
    if ( xhr.status === 200 ) {
        // What do when the request is successful
        console.log( xhr );
    } else {
        // What do when the request fails
```

```
        // What do when the request fails
        console.log('The request failed!');
    }

    // Code that should run regardless of the request sta
tus
    console.log('This always runs...');
};

// Create and send a GET request
// The first argument is the post type (GET, POST, PUT, D
ELETE, etc.)
// The second argument is the endpoint URL
xhr.open( 'GET', 'https://jsonplaceholder.typicode.com/po
sts' );
xhr.send();
```

Browser Compatibility

Works in all modern browsers, and IE7 and above.

JSONP

For security reasons, you cannot load JSON files that reside on a different server. JSONP is a way to get around this issue. Sitepoint explains:[2]

> JSONP (which stands for JSON with Padding) builds on this technique and provides us with a way to access the returned data. It does this by having the server return JSON data wrapped in a function call (the "padding") which can then be interpreted by the browser.

With JSONP, you need to use a global callback function to actually do something with the data you get. When you request your JSON data, you pass in that callback as an argument on the URL (via a query string like this: `?callback=myCallbackFunctionName`). When the data loads, it runs the callback function.

`getJSONP` is a helper function I wrote to handle JSONP requests. Pass in your URL endpoint as the first argument, and your callback function name (as a string) as the second argument.

For testing purposes, we're using JSFiddle's Echo service[3] in the example below.

```
/**
 * Get JSONP data
 * @param  {String}   url      The JSON URL
 * @param  {Function} callback The function to run after
JSONP data loaded
 */
```

```
var getJSONP = function ( url, callback ) {

    // Create script with url and callback (if specified)
    var ref = window.document.getElementsByTagName( 'scri
pt' )[ 0 ];
    var script = window.document.createElement( 'script'
);
    script.src = url + (url.indexOf( '?' ) + 1 ? '&' : '?
') + 'callback=' + callback;

    // Insert script tag into the DOM (append to <head>)
    ref.parentNode.insertBefore( script, ref );

    // After the script is loaded (and executed), remove
it
    script.onload = function () {
        this.remove();
    };

};

// Example
var logAPI = function ( data ) {
    console.log( data );
}

getJSONP( 'http://jsfiddle.net/echo/jsonp/?text=something
&par1=another&par2=one-more', 'logAPI' );
```

Browser Compatibility

Works in all modern browsers, and IE6 and above.

A Helper Library

Atomic[4] is an insanely useful Ajax/HTTP micro-library originally created by Todd Motto[5] and now managed by me. It weighs just 1.5kb minified, and makes working with Ajax/HTTP and JSONP absurdly easy.

You initiate an Ajax request by calling `atomic.ajax()`, and pass in your options as a JavaScript object.

```
atomic.ajax({
    type: 'GET', // This is the default
    url: 'https://jsonplaceholder.typicode.com/posts'
});
```

Atomic uses chained `.success()`, `.error()`, and `.always()` methods to run callbacks when the Ajax call is successful, fails, and either way, respectively.

You can pass two arguments into your callback. `data` is returned data, converted into a JSON object is applicable. `xhr` is the full request, including the response text, URL, status code, and so on.

```
// GET
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/posts'
})
    .success(function (data, xhr) {
        // What do when the request is successful
        console.log(data);
        console.log(xhr);
```

```
        console.log(xhr);
    })
    .error(function () {
        // What do when the request fails
    })
    .always(function (data, xhr) {
        // Code that should run regardless of the request
 status
    });


// POST
atomic.ajax({
    type: 'POST',
    url: 'https://jsonplaceholder.typicode.com/posts',
    data: {
        title: 'foo',
        body: 'bar',
        userId: 1
    }
})
    .success(function (data, xhr) {
        // What do when the request is successful
        console.log(data);
        console.log(xhr);
    })
    .error(function () {
        // What do when the request fails
    })
    .always(function (data, xhr) {
```

```
        // Code that should run regardless of the request
 status
    });


// JSONP

var myCallback = function (data) {
    console.log(data);
};

atomic.ajax({
    type: 'JSONP',
    url: 'https://jsfiddle.net/echo/jsonp/',
    callback: 'myCallback',
    data: {
        text: 'something',
        par1: 'another',
        par2: 'one-more',
        bool: true
    }
});
```

Browser Compatibility

Works in all modern browsers, and IE8 and above.

Working with HTML

How to asynchronously get HTML from another page.

Getting the HTML

You may want to use Ajax to asynchronously get HTML from another page and load it on the current one. To do this, we just need to change the `responseType` of our `XMLHttpRequest()` from it's default (`text`) to `document`.

Hand-Coded

```
var xhr = new XMLHttpRequest();

// Setup our listener to process compeleted requests
xhr.onreadystatechange = function () {
    // Do something...
};

// Create and send a GET request
xhr.open( 'GET', '/page-url' );
xhr.responseType = 'document'
xhr.send();
```

With Atomic

```
atomic.ajax({
    url: '/about/',
    responseType: 'document'
})
    .success(function (data, xhr) {
        // Do something...
    });
```

Replacing the entire page

To replace the entire page with our new HTML, we'll use `innerHTML` to replace the markup inside our `body` element.

Hand-Coded

```
var xhr = new XMLHttpRequest();

// Setup our listener to process compeleted requests
xhr.onreadystatechange = function () {
    // Only run if the request is complete
    if ( xhr.readyState !== 4 ) return;

    // If successful, replace the page content
    if ( xhr.status === 200 ) {
        document.body.innerHTML = xhr.response.body.inner
HTML;
    }
};

// Create and send a GET request
xhr.open( 'GET', '/page-url' );
xhr.responseType = 'document'
xhr.send();
```

With Atomic

```
atomic.ajax({
    url: '/about/',
    responseType: 'document'
})
    .success(function (data, xhr) {
        document.body.innerHTML = data.body.innerHTML;
    });
```

Adding an element to the page

Want to grab content from another page and selectively insert it into the current one? We can do that by using `querySelector` to get our content, and `insertBefore` to add it to the document.

Hand-Coded

```
var xhr = new XMLHttpRequest();

// Setup our listener to process compeleted requests
xhr.onreadystatechange = function () {
    // Only run if the request is complete
    if ( xhr.readyState !== 4 ) return;

    // If successful, replace the page content
    if ( xhr.status === 200 ) {
        // Get our element
```

```
        var elem = xhr.response.querySelector('.some-selector');

        // Get the element to insert it before or after
        var target = document.querySelector('#target-location-selector');

        // Insert it before the target
        target.parentNode.insertBefore(elem, target);

        // OR, insert it after the target
        target.parentNode.insertBefore(elem, target.nextSibling);
    }
};

// Create and send a GET request
xhr.open( 'GET', '/page-url' );
xhr.responseType = 'document'
xhr.send();
```

With Atomic

```
atomic.ajax({
    url: '/about/',
    responseType: 'document'
})
    .success(function (data, xhr) {
        document.body.innerHTML = data.body.innerHTML;

        // Get our element
        var elem = data.querySelector('.some-selector');

        // Get the element to insert it before or after
        var target = document.querySelector('#target-loca
tion-selector');

        // Insert it before the target
        target.parentNode.insertBefore(elem, target);

        // OR, insert it after the target
        target.parentNode.insertBefore(elem, target.nextS
ibling);
    });
```

Putting it all together

To make this all tangible, let's work on a project together. We'll build a photo gallery using the Photos endpoint on JSONPlaceholder.

The starter template and complete project code are included in the source code[6] on GitHub.

Getting Setup

CSS

To get started, we need some lightweight CSS.

We should add `box-sizing: border-box;` to everything to make sure our padding is rendered consistently. We also want to apply a `max-width` of `100%` to our images to make sure they're responsive.

Finally, we'll create a simple grid system. Our `.row` class creates a container for our grids and clears any floats. A `<div>` with the `.grid` class will go around each image. On small screens it's 100% of the width of the page, but on larger viewports, each image will span a fourth of the page.

```
/**
 * Add box sizing to everything
 * @link http://www.paulirish.com/2012/box-sizing-border-box-ftw/
 */
```

```css
*,
*:before,
*:after {
    box-sizing: border-box;
}

img {
    height: auto;
    max-width: 100%;
}

.row {
    margin-left: -1.4%;
    margin-right: -1.4%;
}

.row:before,
.row:after {
    display: table;
    content: " ";
}

.row:after {
    clear: both;
}

.grid {
    float: left;
    padding-left: 1.4%;
    padding-right: 1.4%;
```

```
    width: 100%;
}

@media (min-width: 20em) {
    .grid {
        width: 25%;
    }
}
```

Markup

For body content, I've added a single, empty `<div>` with an ID that we'll hook into to inject our content once we get it from the API.

```
<div id="gallery"></div>
```

JavaScript

First, let's include Atomic to make life a little easier. Then, let's setup a basic Ajax GET request with the JSONPlaceholder Photos endpoint.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // What do when the request is successful
    })
    .error(function () {
        // What do when the request fails
    });
```

Now we can start working with our data.

See what data the API returns

First, let's see what data gets returned from the API.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        console.log(data);
    })
    .error(function () {
        // What do when the request fails
    });
```

If you view the console tab in developer tools, you'll see a 5000 items returned (wow!). Each one includes the following information as a JSON

object:

```
{
    albumId: 0,
    id: 0,
    thumbnailUrl: 'http://someurl.com',
    title: 'The image title or description',
    url: 'http://someurl.com'
}
```

Now that we know the data structure, we can use it to create our markup.

Create the markup

First, let's setup our variables. We want to get our `#gallery` container, and create a placeholder for the HTML content we're going to create.

We also want to make sure the `#gallery` was found. If not, we'll bail.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // Variables
        var gallery = document.querySelector('#gallery');
        var content = '';
        if (!gallery) return;
    })
    .error(function () {
        // What do when the request fails
    });
```

Next, we want to loop through each item in our `data` return with a `for` loop and generate some markup from it.

We'll append each new set of markup to our `content` string by using `+=`. This is a shorthand for `content = content + 'new string'`.

We'll wrap each set of content in a `<div class="grid">`. Inside, we'll add the thumbnail image, and wrap it in a link that points to the fullsized version.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // Variables
        var gallery = document.querySelector('#gallery');
        var content = '';
        if (!gallery) return;

        // Loop through the data and generate our markup
        for (var i = 0; i < data.length; i++) {
            content +=
                '<div class="grid">' +
                    '<a href="">' +
                        '<img alt="" src="">' +
                    '</a>' +
                '</div>';
        }
    })
    .error(function () {
        // What do when the request fails
    });
```

Now that we've got our structure, can add our actual API data to it. We'll use `data[i]` to grab the current item in the loop, and then reference the applicable items in the JSON object.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // Variables
        var gallery = document.querySelector('#gallery');
        var content = '';
        if (!gallery) return;

        // Loop through the data and generate our markup
        for (var i = 0; i < data.length; i++) {
            content +=
                '<div class="grid">' +
                    '<a href="' + data[i].url + '">' +
                        '<img alt="' + data[i].title + '"
 src="' + data[i].thumbnailUrl + '">' +
                    '</a>' +
                '</div>';
        }
    })
    .error(function () {
        // What do when the request fails
    });
```

Inject our markup into the DOM

Now that we've got our markup, we want to inject it into the DOM. We'll use `innerHTML` to do that, wrapping our `content` variable in a `<div>` with the `.row` class.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // Variables
        var gallery = document.querySelector('#gallery');
        var content = '';
        if (!gallery) return;

        // Loop through the data and generate our markup
        for (var i = 0; i < data.length; i++) {
            content +=
                '<div class="grid">' +
                    '<a href="' + data[i].url + '">' +
                        '<img alt="' + data[i].title + '"
 src="' + data[i].thumbnailUrl + '">' +
                    '</a>' +
                '</div>';
        }

        // Inject our content into the DOM
        gallery.innerHTML = '<div class="row">' + content
 + '</div>';
    })
    .error(function () {
        // What do when the request fails
    });
```

We now have our image gallery loaded into the DOM from API content

we've fetched with Ajax!

The finishing touches

There's just two things left to do.

First, let's add a `Loading...` placeholder to our `#gallery` div so that something is displayed on the page while we're waiting for our API data to come back. You could also include a spinner or any other loading indicator you'd prefer.

```
<div id="gallery">
    <p>Loading...</p>
</div>
```

Next, let's add a message in our `.error()` callback that we can display if the API call fails.

We'll again get the `#gallery` and make sure it exists. Then we'll use `innerHTML` to add a message to it.

```
atomic.ajax({
    url: 'https://jsonplaceholder.typicode.com/photos'
})
    .success(function (data) {
        // Variables
        var gallery = document.querySelector('#gallery');
        var content = '';
        if (!gallery) return;
```

```
        // Loop through the data and generate our markup
        for (var i = 0; i < data.length; i++) {
            content +=
                '<div class="grid">' +
                    '<a href="' + data[i].url + '">' +
                        '<img alt="' + data[i].title + '"
 src="' + data[i].thumbnailUrl + '">' +
                    '</a>' +
                '</div>';
        }

        // Inject our content into the DOM
        gallery.innerHTML = '<div class="row">' + content
 + '</div>';
    })
    .error(function () {
        var gallery = document.querySelector('#gallery');
        if (!gallery) return;
        gallery.innerHTML = '<p>Sorry, we could not find
any photos!</p>';
    });
```

Congratulations! You just wrote an Ajax script with vanilla JavaScript.

About the Author

Hi, I'm Chris Ferdinandi. I help people learn JavaScript.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at GoMakeThings.com.
- By email at chris@gomakethings.com.
- On Twitter at @ChrisFerdinandi.

1. https://jsonplaceholder.typicode.com/↩
2. https://www.sitepoint.com/jsonp-examples/↩
3. http://doc.jsfiddle.net/use/echo.html↩
4. https://github.com/cferdinandi/atomic↩
5. https://toddmotto.com↩
6. https://github.com/cferdinandi/ajax-source-code↩