

Forum:[Hack Forums](#)

Thread: [Polymer JS: Attributes and AJAX + building a Bacon-Ipsum placeholder component.](#)

Tasemu - January 11, 2015, 9:17 pm

In the last guide i glossed over what Web Components were and why they were cool. I also briefly covered Polymer, what it did and how to make a very basic polymer component. In this tutorial I want to take a closer look into Polymer Elements and cover some awesome features like attributes (Which basically offer an external api for people to interface with your component) and data-binding (Most should know what this is, basically binding and updating data set within a scoped variables or javascript objects).

In this guide I want to take a more practical approach and talk you through building a finished Polymer Component that gets data from the bacon-ipsun api and uses it to supply your site with some placeholder content. So it will basically look like this:

In your code

```
Code <bacon-ipsun paragraphs="2"></bacon-ipsun>
```

In your Browser

Code Bacon ipsum dolor amet shankle capicola alcatra pork loin cow tongue pork belly jowl spare ribs brisket andouille tenderloin landjaeger pork chop venison. Pancetta doner frankfurter ham venison. Jerky flank kevin doner short ribs alcatra spare ribs ham chuck tongue pastrami chicken salami. Andouille bacon drumstick, tongue leberkas ball tip pork chop ham hock short ribs pancetta biltong shoulder pig meatball. Landjaeger porchetta shankle strip steak spare ribs.

Short ribs beef meatball, ground round tenderloin pancetta ball tip corned beef tongue bacon. Venison rump kevin alcatra pancetta strip steak tongue. Beef turducken biltong ground round meatloaf. Andouille prosciutto jerky, rump ham cow sausage short ribs. Tenderloin venison salami ground round. Strip steak kevin pig pork chop shoulder ground round chicken rump. Drumstick sausage tri-tip bacon, ground round tail chuck turducken meatball swine porchetta.

I think this is a cool little first project because its easy to learn and its useful because it keeps your code nice and clean.

Ok so first things first. Because we want to create a nice and 'sharable' component, we are going to want to follow a few small conventions that make it easier for others to install our module. This is very simple.

Modules are usually installed through Bower. Bower is a package manager for front-end scripts and libraries. Basically if you wanted to install JQuery then instead of going and downloading jquery, extracting it into your JS folder and then including it in your html document; instead with Bower you can just run the following into terminal:

```
Code bower install jquery
```

When you run this, bower will go download jquery and put the file into a folder called

bower_components which will be in the root of your project. So because our Polymer Component will require the polymer library (That will also be downloaded through bower), we need to make sure our path in our component can reach Polymer. The general best practise at the moment is to include Polymer like so:

```
Code <link rel="import" href="../polymer/polymer.html">
```

However we don't really need to worry about stuff like this because the Polymer website has a super quick guide and zip file to get you up and running with a great shareable component scaffold here: <https://www.polymer-project.org/docs/start/reusableelements.html>

Also if you have Yeoman and the [generator-polymer](#) generator for Yeoman then you can just go to your sites directory and run the following in terminal and Yeoman will set up your scaffold for you and you won't need to follow any guide steps!

```
Code yo polymer:seed my-component-name
```

Change my-component-name to whatever you want to call your component. Now your project directory should look a little like this, you will have your element (example: test-element) and all the dependencies. Looking at this, you should see that our link code above will work perfectly here. The link code will come one step out of the test-element folder, then into the polymer folder and finally find polymer.html. This is the best way to structure your dependencies at the moment so make sure to get this bit right or else nobody will be able to use your component without messing around.

Ok so now that we have our base, let's get started making our placeholder text component.

First thing we need for our component is a way to request placeholder text from bacon-ipsum.com so we can use it in our template. For this we will include another component within our component and utilise it for making an ajax request. Components are super easy to use within your component and here is how you do it:

First make sure you are inside your new component folder and not out in the dependencies and then run

```
Code bower install Polymer/core-ajax --save
```

Bower will go and get the dependency for you and drop it into the parent folder with the rest of them. Now open 'my-component-name.html' in your text editor. You should see a template for a blank Polymer Component.

First go down into the Polymer() function below your template tags and change the author to your name, also remove the 'fancy', 'ready', 'fireLasers' and 'sayHello' functions since we won't be needing any of those.

Let's import our core-ajax component. just below the polymer-element tag and before the template tag, paste in the following:

```
Code <link rel="import" href="../core-ajax/core-ajax.html">
```

Make sure it is before your template tags because we do not want our script inside the shadow dom.

Now inside of our template tags paste in this:

```
Code <core-ajax
auto
method="GET"
url="https://baconipsum.com/api/"
params='{"type":"all-meat", "paras":"{{paragraphs}}"}'
handleAs="json"
on-core-response="{{handleResponse}}"></core-ajax>
```

This is our core-ajax component. You can use the attributes to customise the component how you like. There are a few attributes I want to draw attention to. The first is the url, you will see that we are shooting off the ajax request to the bacon-ipsum api. The next is params, notice we are setting the 'paras' parameter to {{paragraphs}}. This is an attribute we will give to our own polymer element which will allow people to choose how many paragraphs of text they would like.

Up in the polymer-element tag, you will see an attribute called 'attributes'. inside the quotation marks, put in 'paragraphs'. Now through the magic of science it will take what people put into paragraphs into your request to the api. like this:

```
Code <my-component-name paragraphs="5"></my-component-name>
```

The other attribute i want to point out in the core-ajax component is 'on-core-response'. This is a function you can create inside your Polymer() function. In this case it is called handleResponse. So lets go down into our script tag and create this function and while we are here, lets set a default value for our paragraphs attribute!

```
Code Polymer({
  author: 'mr awesome',
  paragraphs: 2,
  handleResponse: function (res) {
    var scope = this;
    res.detail.response.forEach(function (element) {
      scope.$.thebacon.innerHTML += (element + "<br><br>");
    });
  },
});
```

Anyway with handle response we are getting the data from the server and passing it into our function as 'res'. Then for each paragraph in the response (they serve each paragraph as an array item), we are getting a node inside our shadow dom with the ID of 'thebacon'.

We havent created this element yet so just below the core-ajax element lets make ourselves this element like so:

```
Code <core-ajax
auto
method="GET"
url="https://baconipsum.com/api/"
params='{"type":"all-meat", "paras":"{{paragraphs}}"}'
handleAs="json"
on-core-response="{{handleResponse}}"></core-ajax>
```

```
<p id="thebacon"></p> // <<< important part here
```

Thats pretty much it for the element guys! So overall it should look pretty much like this:

```
Code <link rel="import" href="../polymer/polymer.html">
```

```

<polymer-element name="bacon-ipsu" attributes="paragraphs">
<link rel="import" href="../../core-ajax/core-ajax.html">

<template>
  <link rel="stylesheet" href="bacon-ipsu.css" />

  <core-ajax
    auto
    method="GET"
    url="https://baconipsum.com/api/"
    params='{"type":"all-meat", "paras":' + {{paragraphs}} + "'"
    handleAs="json"
    on-core-response="{{handleResponse}}"></core-ajax>

    <p id="thebacon"></p>

</template>

<script>
Polymer({
  author: 'mr awesome',
  paragraphs: 2,
  handleResponse: function (res) {
    var scope = this;
    res.detail.response.forEach(function (element) {
      scope.$.thebacon.innerHTML += (element + "<br><br>");
    });
  },
});
</script>

```

</polymer-element>

Now you can register your component with bower and begin creating more components to share with the new component community ecosystem. There is a guide to register your component on Bower here: <http://bower.io/docs/creating-packages/>

Anyway that's all from me today guys. Hope you liked the guide and if you did then maybe come check out my dev blog where I create these guides and share it around! :3

Kina - January 12, 2015, 6:33 pm

Bump, Tas always makes quality tutorials and stuff. :)

Fireworks - January 12, 2015, 7:22 pm

I really don't know what this is. But that's because I'm not on your lvl Tas. Nice tut.

Vip?r - January 12, 2015, 7:46 pm

I got my dick in a toaster, did I do it properly?

Tasemu - January 13, 2015, 7:26 pm

Cheers guys, fuck this section is lazy. Are these tutorials just too advanced or something?

Sakuto - January 14, 2015, 8:37 am

(Jan 13, 2015 08:26 PM)Tasemu Wrote: [\[CITE\]](#) Cheers guys, fuck this section is lazy. Are these tutorials just too advanced or something?

They are complex and so new, not a tons of people are using this kind of technology, and the one who use it don't need tutorials :p

Crake - January 14, 2015, 2:23 pm

Nice walkthrough, though I still think Polymer's never going to really catch on.

Tasemu - January 14, 2015, 2:31 pm

(Jan 14, 2015 03:23 PM)Crake Wrote: [\[CITE\]](#) Nice walkthrough, though I still think Polymer's never going to really catch on.

Really? Its just a tiny layer on top of web components. Web components is definitely going to become the new new of doing web development. That's pretty much globally accepted at this point.

Crake - January 14, 2015, 10:55 pm

(Jan 14, 2015 03:31 PM)Tasemu Wrote: [\[CITE\]](#) Really? Its just a tiny layer on top of web components. Web components is definitely going to become the new new of doing web development. That's pretty much globally accepted at this point.

Exactly. Polymer won't. Web components I'm still iffy about. We'll see... Nice encapsulation, but is it too much? What we really need is native two way data binding.

Tasemu - January 15, 2015, 12:16 am

Agreed, native is always best. However polymer gives 2-way data-binding and a really like declarative way to make web components. I firmly believe web components is the future and after polymer removes its polyfills then it will be be best way to work with web components by far and have a tiny filesize. :)

Crake - January 15, 2015, 12:56 pm

(Jan 15, 2015 01:16 AM)Tasemu Wrote: [\[CITE\]](#) Agreed, native is always best. However polymer gives 2-way data-binding and a really like declarative way to make web components. I firmly believe web components is the future and after polymer removes its polyfills then it will be be best way to work with web components by far and have a tiny filesize. :)

Yeah, but polymer's binding feels too organic and weakly implemented. We need something like Angular to be native I think. The ES6 Proxy class is a good entry way to that.