**Drupal 8 Theming Fundamentals, Part 1**

*by*[*John Hannah*](https://www.lullabot.com/who-we-are/john-hannah)**JUNE 5, 2015**

In this series of posts we’re going to dig into some of the fundamentals of Drupal 8 theming. By the time we’re finished we’ll have a solid understanding of how to apply many of the new tools and techniques in our work. We’ll also have a starter theme we’ll be able to use in our future projects.

We’re going to begin by building the bare minimum required to get our theme working. We’ll create the basic file structure as well as a critical configuration file so that Drupal will recognize our theme and let us enable it.

Before we get started, a brief word on the current state of Drupal 8. At the time of this writing the latest version of Drupal 8 is beta10. That means some of the things below may change as we approach the release candidate. Nevertheless, the large majority of this tutorial should hold up just fine.

**Adding YAML Config**

If you’re familiar with Drupal 7 theming (or module development, for that matter), then you have probably worked with .INFO files before. If not, a .INFO file is found in the root of the theme or module. This file tells Drupal the theme exists and provides other important information. In Drupal 8, the old .INFO files are now gone and have been replaced by [YAML files](http://symfony.com/doc/current/components/yaml/yaml_format.html) - pronounced “yamel” (rhymes with camel).

The move to YAML files is something that you’ll notice throughout Drupal 8. It’s a file format used in the Symfony PHP framework, parts of which are now used in Drupal 8. Fortunately, YAML is pretty straightforward and most folks won’t have trouble making this adjustment.

In a Drupal 8 theme, we create a *.info.yml* file. The naming and placement of this file is very important. It has to include the name of our theme and it has to be saved inside a folder with the same name as our theme. For this project I’m going to name the theme ‘Atlas’. Therefore, the *.info.yml* file will be named *atlas.info.yml* and will be placed inside a folder named ‘atlas’. If your theme has multiple words in its name, they should be separated by an underscore. For example, *MY\_THEME.info.yml*.

The directory structure of Drupal 8 has changed so we won’t be placing this theme in the sites directory as you might expect from Drupal 7 (although that’s still an option if you *really* want to). Instead the file is placed under the ‘themes’ folder that is found in the root of your Drupal 8 installation.

Within the themes folder we can further organize our files into ‘contrib’ and ‘custom’ folders. Contrib themes would be those downloaded from Drupal.org - a base theme, for example. The custom theme folder is where we should place our ‘Atlas’ theme. When we’re done, the folder structure should look like this:

**Adding Required Mappings**

In our *info.yml* file, we’re going to have to create a few key/value pairs called mappings.The four required mappings that we need to create in order for our theme to work are: name, description, type, and core. There are others we will add later, but for now all that we want to do is have Drupal see our theme so that we can enable it. Here’s what the file looks like with the required information added:

name: Atlas

description: A starter theme for Drupal 8.

type: theme

core: 8.x

There are [some rules](http://symfony.com/doc/current/components/yaml/yaml_format.html#strings) regarding the use of quotes in YAML files. If your description has certain characters in it—an ampersand, for example—then you’ll need to put it in quotes. You can use either single or double quotes, just be consistent.

**Enabling the Theme**

There are two ways to enable a theme—by navigating to *admin/appearance* in your browser, or by using Drush. If you choose to use Drush, you’ll need to [update to Drush 8](https://pantheon.io/blog/introducing-drush-8) as it’s required for working with Drupal 8. Although this may be extra work upfront, I highly recommend it as it will save huge amounts of time going forward. Upgrading to Drush 8 may require that you[maintain multiple versions](http://drupal.stackexchange.com/questions/96226/how-to-install-multiple-version-of-drush) of Drush. Many folks won’t run into this, but it’s something to keep in mind.

Once you have Drush 8 installed there is a new command for updating the default theme. To enable the Atlas theme we’ve created, execute the following Drush command:

drush config-set system.theme default atlas

Of course, replace ‘atlas’ in the command above if you’ve chosen a different name for your theme. The changed syntax for updating the default theme is due to the new [configuration management system](https://www.drupal.org/documentation/administer/config) in Drupal 8. For more information on some of the other changes to Drush, [this post](https://www.acquia.com/blog/leverage-drush-7-drupal-8) by Aurelien Navarre is very useful.

**Adding CSS and JavaScript**

Before we really dig into adding our CSS and JS files, let’s get a handle on where things stand. We’ve created our theme and we’ve also enabled that theme. We don’t have anything in our theme besides the *.info.yml* file, so you would expect to see no styling on the page, and mostly, that’s what’s happening. Take a look at the screenshot below from my local machine. I’ve added some sample content using the Devel module for demonstration purposes.

Above, I’ve used a red arrow to point out some styled content where there is padding around a menu item (the bullet has also been removed from the list item). The reason we’re seeing some styling on the page is because Drupal 8 has included the CSS from modules that are providing content to the page.

To identify these files (so that we can remove them) let’s have a look at the source.

Drupal is adding four stylesheets from core. Whether or not you choose to remove these depends on your role. If you’re a front-end developer, then you should probably remove them and keep all of your CSS inside the theme which allows for a separation of concerns and is a best practice.

The good news is that if you do decide to remove them, it’s a snap. Let’s return to our info.yml file so that we can remove these files from our pages. The key that we’re going to be adding is *stylesheets-remove*. Here’s what our *.info.yml* file will look like when we’re done:

name: Atlas

description: A starter theme for Drupal 8.

type: theme

core: 8.x

stylesheets-remove:

- core/assets/vendor/normalize-css/normalize.css

- core/modules/system/css/system.module.css

- core/modules/system/css/system.theme.css

- core/modules/views/css/views.module.css

You’ll notice the new mapping at the end of the file. The key *stylesheets-remove* signals to Drupal that you want to remove one or more stylesheets. It’s followed by what is called a sequence in YAML - basically a list of the CSS files you’d like to remove.

These should be nested below *stylesheets-remove*, indented two spaces and prefixed with a dash. Together these elements are known as a collection in YAML. In standard YAML, the number of spaces the sequence is indented isn’t important, so long as it’s at least 1 space. In Drupal, we use two spaces.

**Note:** Although the stylesheet-remove key will work, it’s due to be phased out in Drupal 9 and replaced by *libraries-override* which will also be available in Drupal 8 and may provide you with more flexibility. For additional information, you can [review this issue](https://www.drupal.org/node/2451411) on Drupal.org. At the time of this writing, however, only *stylesheets-remove* will work.

**Adding Our Libraries**

In order to add our CSS and JavaScript we have to introduce a somewhat new concept for many theme developers—libraries. Don’t worry, this is going to be easy and there is a [good reason](https://www.drupal.org/node/2377397) for the change. We begin by creating a new YAML file in the root of our theme. This file should have the naming convention *THEME\_NAME.libraries.yml*.

We’re adding our files in this way to provide consistency between the way front-end and module developers add asset files like CSS and JavaScript. Let’s take a look at the format of our libraries file which I’ve named *atlas.libraries.yml*

global-css:

css:

theme:

css/style.css: {}

global-js:

js:

js/script.js: {}

dependencies:

- core/jquery

Let’s go over this. The basic structure is referred to as nested mappings and is very similar to what we did for the *info.yml* file. You’ll also notice that I’ve created two libraries—*global-css*and *global-js*. These are names that make sense to me for these global assets, but you can name them anything that you like.

Next we have a key that identifies what type of library we are adding. If you are adding CSS, you’ll next need to include the ‘theme’ key followed by the path to your CSS file(s). I’ve only added one CSS file in this example, but you could add as many as you need (a print stylesheet, for example). The curly braces allow you to add a value to the stylesheet path key - one good example of the usefulness of this is adding a media query for a stylesheet:

css/print.css: { media: print }

This would result in a CSS file being added to the page like so:

<link rel="stylesheet" media="print" href="path/to/theme/print.css" />

For the JavaScript library, we see something a bit different. We have a new key here called ‘dependencies’. In this case I’ve added jQuery to demonstrate what this might look like, but it’s important to note that core doesn’t add jQuery or other JavaScript by default on pages where it’s not needed. This will help keep Drupal websites nice and lean. Also notice that the dependencies need to be added as a sequence (the lines preceded by a dash).

Now that we have our libraries file, let’s add it to our theme. Returning to our *info.yml* file, we now see the following:

name: Atlas

description: A starter theme for Drupal 8.

type: theme

core: 8.x

libraries:

- atlas/global-css

- atlas/global-js

stylesheets-remove:

- core/assets/vendor/normalize-css/normalize.css

- core/modules/system/css/system.module.css

- core/modules/system/css/system.theme.css

- core/modules/views/css/views.module.css

You’ll notice that we’ve added a new mapping with the key of ‘libraries’. Although adding CSS and JavaScript in this way may feel unfamiliar at first, it’s actually pretty easy to get the hang of and will help provide a standard way of dealing with these assets across an entire site.

Let’s take a look at the file structure we have have thus far:

Although not visible in the image above, there is a script.js file in the js folder. One interesting thing is that the reference to these files we’ve just defined will be added to the page even if they don’t exist. Be sure you have added them correctly to avoid 404 errors.

**Adding Libraries to Specific Pages**

Something that often comes up is the need to add CSS or JavaScript to a single page - maybe you’re adding a JavaScript library to the front page for a fancy effect.

Doing this will require that we add a new file to our theme that should be named*THEME\_NAME.theme* and it’s the successor to the *template.php* file in Drupal 7 theming. Let’s add *atlas.theme* in the root of our theme folder.

Next we’ll add the new library to our theme. When we’re done, our *atlas.libraries.yml* file should look something like this:

global-css:

css:

theme:

css/style.css: {}

global-js:

js:

js/script.js: {}

dependencies:

- core/jquery

fancy-effect:

js:

js/fancy.js: {}

We now have a new library defined in our theme, but instead of attaching it globally in our*info.yml* file, we’re going to need to add it via a preprocess function in our *atlas.theme* file (yes, preprocess functions are alive and well in Drupal 8 theming).

Here’s how it would look to add this JavaScript file to our front page via a preprocess function:

<pre>

<code class="language-php">

function atlas\_preprocess\_page(&$variables) {

if ($variables['is\_front']) {

$variables['#attached']['library'][] = 'atlas/fancy-effect';

}

}

In order to get the above function to work, you’re probably going to have to clear your cache. You can either do this in the UI under */admin/config/development/performance* or you can use Drush. The Drush command to clear caches in Drupal 8 is *drush cache-rebuild* or *drush cr* as a shortcut.

**Drupal 8 CSS Style Guidelines**

There are some [new guidelines](https://www.drupal.org/node/1887918) for CSS in Drupal 8. Adoption of the SMACSS file structure is being strongly encouraged and I think this makes a lot of sense, particularly if you are going to be working on projects hosted on Drupal.org or with a team of other Drupal developers. It provides a nice standard that will make it easier for others to understand how to work with your theme.

Another thing you’ll notice from the guidelines is that class naming conventions are also being encouraged. Although it doesn’t explicitly mention [BEM class naming](http://webdesign.tutsplus.com/articles/an-introduction-to-the-bem-methodology--cms-19403) in the docs (at least at the time of this writing), that is essentially what is being recommended. Again, if you’re working on projects that will be hosted on Drupal.org or with a team of Drupal developers, these are good practices to adopt.

The last thing we need to briefly cover concerns a new base theme in Drupal 8 called Classy. Addison Berry does a great job of providing the [backstory](https://drupalize.me/blog/201411/controlling-css-classes-classy-theme-drupal-8) on the origins of this new base theme, but the short version is that it serves to provide additional classes that you may be familiar with from previous versions of Drupal.

Those classes are no longer included by default and since I’m in the group of front-end developers that is glad to see them go, we won’t spend time on how to make use of them. However, it’s useful to know they are available if you’d prefer to add them to your work. Check out Bartik theme (the default theme in Drupal 8) for an example of how to set up Classy as a base theme.