First Edition

Aurora

a sass+compass based html5 base theme for drupal

Sam Richard
Front End Developer, WorkHabit

Welcome

Thank you for using Aurora, the Sass+Compass powered, content driven fluid grid Drupal theme based on tried-and-true typographical knowledge and designed to allow for a content driven responsive design. Aurora is based off of the Aura Compass Extension and is required in order to use this theme, as is the current version of the Compass gem and Sass version 3.2, or the prerelease version of Sass if 3.2 is not yet out.

Did you know that there is an entire suite of modules aimed at making both Aurora and all other HTML5 and Responsive themes better called Borealis? The Borealis Suite includes two modules right now, Borealis Semantic Blocks which adds the ability to have truly semantic HTML5 blocks, and Borealis Responsive Images, which allows you to make existing Image styles responsive in an easy to use manner That Just Works™. Soon, it will also include a helper module specifically for easing the creation of Aurora subthemes called Aurora Borealis. Keep an eye out for it.

Aura, Aurora, and Borealis are the brainchild of Sam Richard, a Front End Developer at WorkHabit. He is Snugug throughout the Internet, and can be reached at sam [at] workhabit [dot] com, @Snugug on Twitter, or Snugug in many of the Drupal IRC rooms. Feel free to reach out if you have any questions.

Installing Sass, Compass, and

1

Sass, Compass, and Aura drive Aurora, and as such, need to be installed and used in order to be used. We are going to be using the Command Line, but don't be scared, it'll be easy!

Let's get you started!

Mac Installation

Easy Peasy Lemon Squeezy!

Mac users, you have it off easy! All OSX installs come with Ruby Gems installed by default, so it's super easy to install Sass, Compass, and Aura. Open up your Terminal (either search for it using Spotlight or an application launcher like Alfred, or by opening it manually, by default located in Macintosh HD->Applications->Utilities) and typing in the following commands, pressing return after each command. You will be asked to type in your password the first time you do this:

```
sudo gem install compass
sudo gem install sass --pre
sudo gem install compass-aura
```

That’s it! You’re done!

Linux/Unix Installation

Building from Source

Unix and Linux do not usually come with Ruby/Ruby Gems installed, and as such you're going to need to install them yourselves. If you are not on a system that supports apt-get, follow the instructions from Ruby Gems:

Apt-Get *ALL* The Things!

If you are on a Unix/Linux system that supports apt-get, the instillation of Ruby and Ruby Gems is fairly easy. Open up your command line and apt-get the following and write the gem's path to PATH:

```
sudo apt-get install ruby-full build-essential
sudo apt-get install rubygems
export PATH=/var/lib/gems/1.8/bin:$PATH
```

Installing Sass, Compass, and Aura

Once you have Ruby Gems installed, or if you had Ruby Gems installed already, installing Sass, Compass, and Aura is very easy! Pop back into your command line, and type the following:

```
sudo gem install compass
sudo gem install sass --pre
sudo gem install compass-aura
```

Windows Installation

Get a Drink, We're Gonna Be Here a While

Getting Sass, Compass, and Aura set up on a Windows computer is a little bit harder than on a Mac or Linux/Unix because Ruby Gems is a little harder to get running. In addition to installing Sass, Compass, Aura, and Ruby Gems, we are also going to install Cygwin to make working in the command line easier and make the commands throughout this book work for you as well.

Installing Cygwin

Cygwin gives your Windows system a Unix-like command line interface, which will make your life developing on Windows easier and the commands used throughout this manual work on your computer. Cygwin is free and open source project. See the official documentation on how to install and use Cygwin.

Installing Ruby Gems

Installing Ruby and Ruby Gems on a Windows computer is surprisingly not too hard, simply go to Ruby Installer, download the current version, and follow the setup wizard.

Installing Sass, Compass, and Aura

Once you have Cygwin and Ruby installed, installing Sass, Compass, and Aura is very easy! Open up Cygwin and type the following:

```
gem install compass
gem install sass --pre
gem install compass-aura
```

Getting To Know Aurora

2

What makes Aurora, Aurora? How does it do the crazy things it does? Learn all this and more in here!

HTML5 Base Theme

The HTML5 Project

The HTML5 Project is "…the the official back-port of the theme-related work done in the Drupal 8 HTML5 initiative" and is the foundation of Aurora. The HTML5 project is maintained by Jacine Luisi, the Drupal 8 HTML5 Initiative owner, with contributions by many of those involved in the Drupal 8 HTML5 initiative, including myself. Many of the Core templates and HTML5 updates have been converted and implemented in this project already, with the remaining being added as an official solution presents itself.

But There's No Stable Release!

While it is true there is no stable release for the HTML5 project, and the project itself specifically says not to use, I promise it's okay. The reason there is no official release is because the project is a moving target and the project is designed to be forked from. That being said, my philosophy for Aurora is that, come Drupal 8, it should be as easy as removing the base theme requirement for the HTML5 project for this to be D8 ready, and as such, I'm *not* forking the HTML5 enhancements and am instead straight using the project. Additionally, as I am actively helping Jacine keep the HTML5 project up-to-date, I know the changes that are going into that project and can pivot Aurora if need be. As such, **totally use it;** at least as in conjunction with Aurora.

What About The Other Stuff?

There will be no Core Templates in Aurora. If a Core template hasn't been HTML5-ified and put into the HTML5 project, it will not be part of Aurora. I am trying to keep Aurora as close to Drupal 8 HTML5 as possible, partially to make the upgrade path that much easier, and partly because I'm not the smartest person in the room and would prefer to keep a consensus amongst the Drupal HTML5 community.

That being said, Aurora does make some enhancements through template.php (see the next section, Aurora Enhancements, for a full listing of what Aurora does on top of HTML5 Project) to what is available through HTML5 and, if something doesn't tickle your fancy, you are always welcome to create a template in your subtheme to your liking, or find a module that can help such as *Fences* for field templates, *Elements* for HTML5 form elements, and *Borealis* for semantic block templates and responsive images.

Aurora Enhancements

Enhancements Overview

Modernizr

To quote Modernizr’s Website:

> ***Modernizr*** *is an open-source JavaScript library that helps you build the next generation of HTML5 and CSS3-powered websites.*

At its very basic, Modernizr checks for browser capabilities and adds classes to the HTML tag of the page per HTML5/CSS3 capability of the current browser, thus allowing for easy progressive enhancement by targeting those classes with your custom JavaScript and CSS. Aurora is using the Modernizr build from HTML5 Boilerplate, providing us with Modernizr, an HTML5 Shiv, and Modernizr.load, also known as yepnope.

Footer JavaScript

Aurora, by default, moves all JavaScript that is loaded into your page using drupal_add_js to the *bottom* of the page in an effort to increase page load time and prevent JavaScript from blocking the rendering of your pages. Because functionality of Modernizr is needed as a bare minimum for our pages to work correctly, it is the only piece of JavaScript that, by default, gets loaded in the header of our page. If you would like your JavaScript to be printed in the header, make make your JavaScript’s scope *aurora_header.* See the drupal_add_js documentation for more information about JavaScript scope.

Compressed Theme CSS

Because of the use Sass+Compass in theming, by default, the CSS generated by your subtheme will be compressed. See Configuring Your Output in Chapter 3: Creating Your Subtheme if you would like to change the way your CSS is output. If you are working in a team and committing to a shared version control system such as Git where more than one person will be making changes to the Sass and compiling CSS, I would suggest changing the output to :expanded to alleviate potential merge conflicts.

No Base Theme CSS

Thanks to the cunning use of Sass+Compass and Sass's @import functionality, Aurora ships with no default CSS that needs to be overridden (other than the CSS generated for the layout, which is controlled by your variables. See Basic Setup in Chapter 4: Working with Aura for a full explanation of the basics of working with Aura's Variable System).

Viewport Support

The Viewport tag is added to the page's header allowing for responsive designs to work properly on mobile devices. The default implementation suffers from the iOS Rotation bug and probably will be changed in the future.

Chrome Frame

Aurora will add Chrome Frame to all supported versions of Internet Explorer, which, at the time of this writing, are Internet Explorer 6, 7, 8, and 9.

Internet Explorer Render Engine

There are times when Internet Explorer will not use the most up-to-date render engine when rendering. A tag is added to the header of all pages that will tell Internet Explorer to use the most up-to-date version of its render engine or, if Chrome Frame is installed, use Chrome Frame.

Module Support

List Of Supported Modules

Views

More semantic views-view.tpl.php, views-view-field.tpl.php, and views-view-fields.tpl.php are included make for cleaner views. Read more about the Views module on the Views Drupal Project Page.

Views Slideshow

A more semantic views-slideshow.tpl.php is included that also includes the ability to hide the wrapping skin tag for views slideshows (by default, it is set to true). Read more about the Views Slideshow module on the Views Slideshow Drupal Project Page.

FlexSlider

My personal preferred responsive slideshow, support for the FlexSlider Views Slideshow module is built in to Aurora, specifically hiding the skin for Views Slideshow which is unneeded for FlexSlider, and hard removing the default wrapping div if not already removed. Read more about FlexSlider on the FlexSlider Home Page and more about the FlexSlider module on the FlexSlider Drupal Project Page.

Creating Your Subtheme

3

This is great, how do I get started you say? Well look no further!

Anatomy of the Starterkit

Files In The Aurora Starterkit

1. starterkit.info
2. style.scss
3. _base.scss
4. _variables.scss
5. _mixins.scss
6. _functions.scss
7. _extendables.scss
8. config.rb
9. style.css

starterkit.info

starterkit.info is the generic .info file for your subtheme. It has a generic name and description, sets the Drupal Core to use, the base theme, the CSS file to be loaded in, and the default regions available to you.

style.scss

style.scss is a **Sassy CSS** formatted Sass file located in the *sass* folder of the starterkit. Aurora is built using an object-oriented approach to Sass **partials**, and as such, style.scss should not hold any styles itself, but rather links to partials that, when the file gets compiled, will pull in selectors from the various partials. This should also be the only file that gets compiled, compiling into style.css. The @import statements you see in this file *are not CSS imports* but rather compile time imports, meaning they do not have any effect on CSS load time.

_base.scss

_base.scss is the first partial you will come across in the folder *sass/partials/global* and is one of the most important partial files in the starterkit as it pulls in the master Aurora partials that actually do the magic of setting up your layout. _base.scss is well documented and the instructions in there should be followed exactly or it is possible that your theme will not work correctly.

_variables.scc

_variables.scss is the file where all of the Sass **variables** you want to use in your theme should be stored, and is also where all setup variables should go to set up your layout. See Basic Setup in Chapter 4: Working with Aura for a full explanation of the basics of working with Aura's Variable System.

_mixins.scss

_mixins.scss is the file where all of the Sass **mixins** you want to use in your theme should be stored.

_functions.scss

_functions.scss is the file where all of the Sass **functions** you want to use in your theme should be stored.

_extendables.scss

_extendables.scss is the file where all of the Sass **extendables** you want to use in your theme should be stored.

config.rb

config.rb is your **Compass configuration file**. It holds configuration information for Compass and generally only needs to be edited if you want to add a Compass extension to your theme or change the output style.

style.css

This is the output CSS file. You should consider this file compiled code and *never ever ever* make any changes to it directly as any changes made there *will be overridden next time the CSS file gets compiled*. Best to just ignore it.

A New Theme Is Born!

Creating And Configuring A Subtheme

Copypasta the Starterkit

The first step in creating a new Aurora subtheme is to copy the starterkit to your themes directory, which should result in you having the following two directories: *sites/all/themes/aurora* (where the base theme lives) and *sites/all/themes/starterkit* (your new starterkit). Rename your starterkit directory and starterkit.info to the name of your subtheme (remember, no spaces in folder or file names!), open config.rb and change starterkit in the http_path variable to your new subtheme's name, and finally open your subtheme's new .info file and change the name to your subtheme's name and, if you'd like, the description as well, and tada! A new theme is born!

Configuring Your Variables

Once you have your subtheme set up, you're going to need to set up your variables to get your theme's layout correct. Your variables go in *sass/partials/global/_variables.scss*. See Basic Setup in Chapter 4: Working with Aura for a full explanation of the basics of working with Aura's Variable System.

Configuring Your Output

If you would like to change the output of CSS, it's as simple as changing the output_style in *config.rb*. By default, the output style is :compressed, but you can change it to any of the following: :expanded, :nested, :compact, or :compressed. If you would like your output CSS to have comments above each selector telling you what line of which partial it comes from, set line_comments to true *config.rb.*

Available Regions

There are seven regions available to you out of the box in Aurora: Header, Help, Highlighted, Content, Primary Sidebar, Secondary Sidebar, and Footer. The CSS for the sidebars is generated by Aura's Sidebar Variable, and their positioning can change depending on how you configure it, thus, there is no “Left Sidebar” or “Right Sidebar” as these are determined by your Sidebar Variable.

Working With Aura

4

Aura is the framework that powers Aurora. Learn how to harness the power of Aura to set up your theme!

Basic Setup

Basic Variables

The Philosophy of Aura

Aura is built on some basic principles of typography, specifically around the area where the written word goes called the **meas-ure** and how the measure relates to font size and **leading** (line height). With this knowledge in hand, and a little magic, Aura will generate a fluid, em based grid with breakpoints where the measure falls below an ideal reading length, at which point, it will make the font size smaller, this increasing number of letters per line, and you're back in business!

That being said, what I dub the *font scale* method is a little bit radical for many people, so if you would like to design to a more recognizable grid, Aura also lets you create a grid based on ei-ther your measure without font scaling, or based on a known max width.

Basic Grid Setup

There are two main variables that do the most to determine what your grid is going to look like, the *total-cols* variable and the *main-content-cols* variable. By default, these are set to 12 columns across with a 9 column main content span (this is your measure). Changing them is as simple as adding the following variables to your _variables.scss file:

```
$total-cols: 16;
$main-content-cols: 13;
```

In the above example, I'm changing the grid from a 12 column grid to a 16 column grid and changing the measure's span to 13 columns. The Aurora base theme provides a variable to allow you show and hide a visual representation of your columns in your browser by toggling the *debug* variable which, by default, is set to true:

```
$debug: false;
```

Sidebar Setup

Sidebar setup is a little bit more complicated than the basic grid setup as sidebars are a little more flexible than the measure and overall grid. Aura has built in support for two sidebars as I've found that the most common on website. By default, Aura generates CSS for one sidebar on the right, which looks like this:

```
$sidebars: 1, 'right';
```

If I wanted to change that to a left sidebar, it would be as easy as changing 'right' to 'left':

```
$sidebars: 1, 'left';
```

The way the width gets calculated for this setup is by taking the total number of columns and subtracting from it the number of columns the main content spans. Using our above example, either of these sidebars would wind up having a width of 3 columns. For Aurora's purposes, this sidebar is the Primary Sidebar, and the Secondary Sidebar does not get a width associated with it. But what if you want two sidebars? Well that's as easy as changing the 1 to a 2:

```
$sidebars: 2, 'left';
```

What this now does is calculate the width of two sidebars and, in this example, has both of them sit to the left of the measure. It then calculates the width the same way as before, dividing the remaining number of columns in two. But wait, you say! That works fine for an even number of left over columns, but we have an odd number of left over! This is where Primary Sidebar and Secondary Sidebar come in to play. What Aura does is put the larger "half" of the left over columns in the Primary Sidebar and the smaller "half" of the left over columns in the Secondary Sidebar. With that in mind, our Primary Sidebar would have a width of 2 columns and our Secondary Sidebar would have a width of 1 column.

Now say you'd like your sidebars on each side of your measure? Well that's easy! Just change 'left' to 'both'!

```
$sidebars: 2, 'both';
```

Thus ends the sidebar calculation portion of the Sidebar section. What if you know how many columns you would like your sidebar to span? Well for a single sidebar it's as easy as adding another parameter to the *sidebars* variable with the number of columns you'd like it to span:

```
$sidebars: 1, 'left', 4;
```

Our one left sidebar will now be span 4 columns. But wait, you say! That now means the total number of columns we have exceeds what we set *total-cols* to be! True; Aura assumes you know what you're doing and instead of throwing nasty errors or freaking out, it simply adjusts *total-cols* to be correct!

Now what if we have two columns and want them each to span the same number of columns, well that's easy! It's the same syntax as above, but you increase the column count to two!

```
$sidebars: 2, 'left', 4;
```

This will give you two sidebars to the left of the measure, each spanning 4 columns. Want them on the right? Change 'left' to 'right'!

```
$sidebars: 2, 'right', 4;
```

Same goes for both!

```
$sidebars: 2, 'both', 4;
```

Our *total-cols* is now 21; 13 column span for the measure, 4 column span for each sidebar. Easy peasy.

Let's not assume that you know exactly how many columns you'd like both your Primary Sidebar and Secondary Sidebar to span, and that that's a different number. Well, couldn't be easier, just add the span for the Secondary Sidebar after the span you already have in, and that first one becomes the span for the Primary Sidebar. For instance, you can do the following:

```
$sidebars: 2, 'both', 4, 3;
```

Which will create sidebars on both the left and right side of the measure, with the left sidebar (the Primary Sidebar) spanning 4 columns and the right sidebar (the Secondary Sidebar) spanning 3 columns. Works with left and right too, with the left most column always being the Primary Sidebar and the right most being the Secondary Sidebar.

Scaling Variables

There are three “scaling” variables that can be called in the basic Aura setup: the *body-font-size* variable, the *font-scale* variable, and the *obj-scale* variable.

The *body-font-size* variable controls the median font size for when *font-scale* is true and the actual body font size when *font-scale* is false. By default, *body-font-size* is set to 16px, but it can be set to anything you’d like, in any measurement you’d like; percent, em, pt, or px. Whatever you choose will be converted into em for responsive awesomesauce. Say I wanted to change *body-font-size* to 13pt I could do the following:

```
$body-font-size: 13pt;
```

The next scaling variable is *font-scale*. Font scale is a boolean, by default set to true, that scales your body font based on ideal readability, the raison d'être of Aura, but if you’re not comfortable with how radical it is, simply set *font-scale* to false:

```
$font-scale: false;
```

The final scaling variable is *obj-scale*. Object scale determines whether or not to have all objects (images, video, iframes, objects, etc…) scale with their parents, *i.e.,* giving all of those tags the CSS property *max-width: 100%*. By default this is set to true, but you can disable it by setting it to false:

```
$obj-scale: false;
```

Known Width Grid

Unlike just disabling *font-scale*, which will still build your grid based on your measure, you can still define a known max width for your grid by setting the *width* variable. Doing so will build a grid based on your *total-cols* variable, but the size of each column will be determined by your known max width as opposed to based on your measure. Columns added sidebars will stay within the grid max width. Setting a width disables *font-scale*. To set a known width, set the *width* variable:

```
$width: 960px;
```

Please note that this still build a fluid grid and not a fixed size grid.

Advanced Setup

The Neckbeard Stuff

Why Would You Touch This Stuff?

The next section presents advanced setup options for Aura that you only really need to touch if you feel like getting in to the nuts and bolts of how Aura calculates everything for you. You shouldn't have much reason to touch most of this stuff, but if you want to, it's here.

Responsive Ratio

The *responsive-ratio* variable determines the ratio at which breakpoints are calculated, expressed as a floating point number. By default, if *font-scale* is true, the responsive ratio is 'major thirds' as defined by the Modular Scale Compass extension, if set to false, the responsive ratio is 'golden' ratio as defined by Modular Scale. Feel free to set this to anything you'd like, but do test it.

```
$responsive-ratio: $golden;
```

Gutter to Column Ratio

The *gutter-to-col* variable determines the ratio of gutters that make up one column. The default is 1 column equals 4 gutters, or a 1:4 ratio. This is the default that 960gs works off of, so I figured it would be good for Aura. It is presented as a ratio, with the gutter first and the column second, separated by a comma. To change it, simply change the ratio:

```
$gutter-to-col: 2, 5;
```

Alphabet Count

The *alphabet-count* variable sets how many characters there are in the alphabet you are working with, and therefore the the length of your measure (an ideal measure is between 2 and 3 alphabets long). By default, it is set to the English alphabet's 26 characters, but can be changed:

```
$alphabet-count: 28;
```

Measure Width

The *measure-width* variable sets the number of characters the measure should be, generally defined as the *alphabet-count* times a multiplier, but can in theory be set directly:

```
$measure-width: $alphabet-count * 2.5;
```

Line Hight Addition

If *font-scale* is disabled, *lh-addition* is the amount that will be added to the converted *body-font-size* to determine leading. By default, this is set to .3em, and must be set in em in order work.

```
$lh-addition:.5em;
```

Compiling Your CSS

All Dressed Up With Nowhere To Go

All Dressed Up With Nowhere To Go

Once you have all of your variables set and are ready to start theming your site, you're going to need to get Compass compiling your Sass into CSS for you! There are two approaches, either through an application, or straight through the terminal. The former approach is better for those who are afraid of the terminal but offers less power. The terminal approach can be a little bit daunting at first, but is the more powerful way to start compiling your CSS.

The Application Approach

There are a few applications that will compile Sass into CSS for you without the hassle of the terminal, but I'm going to highlight two here, one that I personally use (and I promise I'm not being paid to advertise it), and one that many people in the community use and I've heard nothing but good things about; they are **LiveReload** and **Scout App**.

LiveReload is a nifty little server app that, at its core, listens for changes in a folder (like a change in a Sass file in your theme) and refreshes the tab you are in with the new changes, allowing for a great **Design In Browser** workflow. LiveReload is available for both Windows and Mac computers and will compile not only Sass+Compass but Less, Stylus, CoffeeScript, IcedCoffeeScript, Eco, SLIM, HAML, and Jade as well. I personally use LiveReload for its Design in Browser capabilities and compile my CSS through the terminal, so its flexible like that. LiveReload uses your installed Ruby gems, making it possible to use LiveReload to compile your CSS even though we need the --pre version of Sass for Aura.

Scout App is a free cross-platform application for Windows and Mac computers that has a self-contained environment to run Sass+Compass and compile your CSS for you, but it does not have the browser refreshing like LiveReload has. Additionally, I am not sure if it works with the --pre version of Sass, which is needed by Aura, because it is a self-contained environment, but once Sass 3.2 is officially released and Scout App gets updated, there should be absolutely no issues using it.

Unfortunately Unix/Linux friends, but I'm not aware of an Application based approach for you. That being said, you should be comfortable enough in the terminal to be fine doing it that way, I mean the terminal *is* an application after all, am I right?!

The Terminal Approach

This is where we jump back into the command line. Open up your Terminal on your Unix/Linux or Mac computer or Cygwin on your Windows computer and navigate to your subtheme's directory. This list of Basic Terminal Commands will help you navigate to your subtheme.

```
  > cd ~/Sites/drupal/sites/all/themes/aurora-
subtheme
```

Make sure you are in the same directory as your config.rb file by listing the contents of your current directory:

```
  > ls
  config.rb css sass aurora-subtheme.info
```

Great, I see config.rb. To start Compass watching your subtheme for changes, simply enter compass watch"

```
  > compass watch
  >>> Compass is polling for changes. Press Ctrl-C to
Stop.
```

That's it! Whenever you save a Sass file in your theme, Compass will see that, try and compile it and, if successful, overwrite your CSS file and, if not, tell you the error and have you go fix it. To stop Compass from watching for changes, either press Ctrl+C on your keyboard or close your terminal window.