

Sass and Compass IN ACTION

Wynn Netherland
Nathan L. Weizenbaum
Christopher M. Eppstein



MEAP



MANNING



**MEAP Edition
Manning Early Access Program
Sass and Compass in Action version 5**

Copyright 2011 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Part 1: Introducing Sass and Compass

1. Sass and Compass make stylesheets fun again
2. Basic Sass syntax
3. Getting to know Compass

Part 2: Real-world stylesheets

4. CSS grids without the math
5. Eliminate the mundane using Compass
6. CSS3 with Compass
7. From prototype to production
8. High-performance stylesheets
9. Wrangling IE and legacy browsers

Part 3: Writing Compass frameworks and plugins

10. Scripting with Sass
11. Case study: Sencha Touch
12. Compass community plugins
13. Building a personal framework
14. Sharing a plugin with others

Appendixes

- A. Installing Sass and Compass
- B. Configuration options
- C. Compass command line
- D. Debugging tips and tools

Sass and Compass make stylesheets fun again



This chapter covers:

- Getting started with Sass and dynamic stylesheets
- Writing stylesheets more efficiently with Sass features
- A quick introduction to Compass
- Compass solutions to real-world stylesheet challenges

Sass is an extension of CSS3 that lets you create better stylesheets with less effort. Sass frees you from repetition and gives you real tools to be creative. Since you can implement changes much faster, you're now free to take risks in your designs. Your stylesheets can now keep pace with changing colors and changing HTML markup, all the while producing standards-based CSS you can use in any environment. Sass is built in Ruby, but unless you want to hack on the language itself, you need not care.

Throughout this book we'll speak to two sets of readers, hoping to find some common ground between each camp. If you find yourself in both groups, even better.

To our web designer friends:

You're the guy or gal who has all the Adobe app keyboard shortcuts memorized. You choose complementary colors based on RGB values alone. You may or may not sport a pair of dark-rimmed glasses, but chances are you start your day with coffee or tea and the latest from Smashing Magazine. By your own admission, you know enough jQuery to be dangerous and don't know why your developer friends chuckle when you talk about CSS as a language.

We're going to set you free from the tedious and let you do what you do best - be creative. We know you have opinions on resets, typographic scales, color

palettes, and layouts. We're going to show you how to create stylesheets faster with less repetition. You'll start doing less in graphics software and more in your stylesheets.

To our front-end developer pals:

You take pride in your ability to slice-and-dice a Photoshop comp into semantically sound HTML and CSS, but there's a problem. Your server templates are very DRY because you *Don't Repeat Yourself*, but your stylesheets are as soggy as a doorbell-interrupted Raisin Bran breakfast. As the project grows, you also find that organizing your stylesheets is a challenge. If only you could author stylesheets in the same way you write the other code in your software project - with variables, reusable parts, and control flow. Take heart, have we got a project for you.

In this chapter we'll look at powerful Sass features such as nested rules, variables, mixins, and selector inheritance that free you from mindless repetition and let you focus on your design instead of your styles. If you don't already have Sass installed, go ahead and jump to Appendix A and follow the steps outlined there. If you're reading this at the coffee shop on your iPad, you can still run these basic examples online at the Sass web site. <http://sass-lang.com/try.html>

1.1 Getting started with Sass

Before we jump into some examples, it's important to nail down some keys to success with Sass. It's not a silver bullet or pixy dust. Sass won't instantly help your color, typography, or layout choices, but it can help you implement your ideas faster, with less friction.

1.1.1 From CSS to Sass

If you're skilled in creating CSS, you'll find the onramp to using Sass a short one. Sass focuses on *how* to create awesome stylesheets, not what goes into them. Later in the book, we'll cover tools like Compass that provide you with CSS best practices, but ultimately you'll benefit from this book if you've got a firm grasp of CSS. As with anything in computing, *Garbage In, Garbage Out*. If you need a CSS primer, you might check out another Manning title, *Hello! HTML5 and CSS3*.

Sass supports two syntaxes:

SCSS, or *Sassy CSS*, was introduced in Sass 3.0 and is a superset of CSS3. It is chock-full of familiar braces and semicolons.

```
h1 {color: #000; background: #fff}
```

Sass got its start as part of the Haml project (<http://haml-lang.com>) before it was split out as a standalone project on its own. The original *indented syntax* was inspired by the Haml project and is whitespace aware instead of requiring braces and semicolons:

```
h1
  color: #000
  background: #fff
```

Sass will continue to support both syntaxes. You can even mix and match each syntax within the same Sass project (just not within a single file). It's important to choose a syntax that's right for you and your team. If you work in a Python or Ruby/Haml environment, perhaps the whitespace-aware indented syntax will fit nicely. If your team deals with outside design agencies, then Sassy CSS provides a lower barrier to entry.

In this book, we'll deal with the new SCSS syntax (unless otherwise noted). In addition to sound CSS skills and a grasp of Sass syntax, it's important to take a dynamic view of stylesheets.

1.1.2 *Think dynamic*

Outside of basic brochure sites, who really writes much static HTML anymore? We take our HTML and we carve it up for our blog engine, CMS, or application framework to *preprocess*, mixing markup and dynamic content. So *why do we still write static stylesheets?* We'll take the concepts you take for granted in creating markup and apply them to creating stylesheets. The key thing to remember is that while Sass lets you write stylesheets in a dynamic fashion, the output is still 100% pure static CSS. Once you're working with dynamic stylesheets, we can now listen to that inner voice that keeps shouting *Don't Repeat Yourself*.

1.1.3 *Don't Repeat Yourself*

Sass gives stylesheet authors powerful tools that remove the tedium from many CSS tasks we do over and over and over. Many features embrace the Ruby on Rails philosophy of *Don't Repeat Yourself*, letting you *DRY* up your stylesheets. As you create your stylesheets, repetition should be a red flag. Constantly ask yourself *how can I work smarter, not just harder?* In the next few sections, we'll show you how to let Sass squeeze more reuse out of your stylesheets.

1.2 Hello Sass: *DRYing up your stylesheets*

So, we've been harping on DRY-DRY-DRY up to this point. So what does a soggy stylesheet look like? Consider the following CSS:

```
h1#brand {color: #1875e7}

#sidebar { background-color: #1875e7}

ul.nav {float: right}
ul.nav li {float: left;}
ul.nav li a {color: #111}
ul.nav li.current {font-weight: bold;}

#header ul.nav {float:right;}
#header ul.nav li {float:left;margin-right:10px;}
#footer ul.nav {margin-top:1em;}
#footer ul.nav li {float:left;margin-right:10px;}
```

Even in this extremely simplified example, the duplication is apparent. What happens if the marketing team wants to tweak that lovely shade of blue from #1875e7 to #0f86e3? Sure two occurrences is manageable, but when it's a dozen or more across several stylesheets, find-and-replace seems a bit archaic, don't you think? Eight instances of `ul .nav` in a ten line stylesheet also seems a bit excessive.

In the next few sections, you'll discover a cool breeze of syntactic sugar that will DRY up this stylesheet and blow you away including variables, mixins, nested selectors, and selector inheritance.

1.2.1 Reuse property values with variables

Are you using search-and-replace to swap hex code values and manage color palette changes in your stylesheets? With Sass, you can assign values to *variables*, and manage colors, border sizes, and virtually any stylesheet property value in a single location:

```
$company-blue: #1875e7;

h1#brand {
  color: $company-blue;
}

#sidebar {
  background-color: $company-blue;
}
```

Sass variables start with the \$ symbol and can contain any characters that are

also valid in a CSS class name, including underscores and dashes. In this simple example, if we want to tweak the the shade of blue, we can update it in one spot and the rest of our stylesheet falls in line.

If you come from a development background, variables should feel natural. If you're coming to Sass from a design background variables may seem intimidating at first glance. However, they are really nothing new. You already use named values in CSS such as `blue`, `green`, `inherit`, `block`, `inline-block`, `serif`, and `sans-serif`. Think of variables as your very own special values. Next up, using nested selectors to create deep descendant CSS selectors with less typing.

1.2.2 Write long selectors more quickly with nesting

Did you ever hear about the Texan who went to work for the state painting dashed center lines on the highway? He was a top performer his first week, painting ten miles of road. Production tailed off quickly, however, as he covered five miles in his second week and only two in the third. When he rounded out the last week of the month with only a single mile, his supervisor asked him what seemed to be the problem. "Well," the worker remarked, "it keeps getting farther and farther back to the bucket."

That's exactly how it can feel working with deep descendant CSS selectors. Consider the following CSS:

```
ul.nav {float: right;}
ul.nav li {float: left;}
ul.nav li a {color: #111}
ul.nav li.current {font-weight: bold;}
```

Sass lets us *DRY* that up a bit. Find the file `1.1.2.nesting.scss` in the code examples folder for Chapter 1 or create your own by saving a text file with the following contents.

```
ul.nav {
  float: right;

  li {
    float: left;
    a {
      color: #111;
    }
    &.current {
      font-weight: bold;
    }
  }
}
```

```
}
}
```

From your terminal, run the `sass` command and pass it the path to the file:

```
sass 1.1.nesting.scss
```

You should get the following CSS results in your terminal output:

```
ul.nav {
  float: right; }
ul.nav li {
  float: left; }
ul.nav li a {
  color: #111; }
ul.nav li.current {
  font-weight: bold; }
```

Other than a bit of formatting differences, that's the same CSS with which we started. (Don't sweat the format just yet. We'll discuss more about Sass's output options a bit later.)

Using Sass, we can *nest* rules and avoid duplicating the same elements in our selectors. Not only does this save time, the added benefit is that if we later change `ul .nav` from an unordered list to an ordered list, we only have one line to change. This is especially true with the last selector in the example. The `&` is a *parent selector*. In this case `& .current` evaluates to `li .current`. If the markup were to change to using the `current` class on some other element, this line in the stylesheet would just work. Now that we've seen how to reuse values with variables and write longer selectors with nesting, let's put the ideas together and look at Sass mixins.

1.2.3 Reuse chunks of style with mixins

Variables let you reuse *values*, but what if you want to reuse large blocks of rules? Traditionally in CSS, as we see duplication in our stylesheets, we factor common rules out into new CSS classes:

```
ul.horizontal-list li {
  float: left;
  margin-right: 10px;
}

#header ul.nav {
  float: right;
}
```

```
#footer ul.nav {
  margin-top: 1em;
}
```

We then need to give our `ul.nav` elements an additional class of `horizontal-list`. This works just fine, but what if we wanted keep our classes a bit more semantic and still get the reuse?

Let's open or create `1.1.2.mixins.scss`, our second example:

```
@mixin horizontal-list {
  li {
    float: left;
    margin-right: 10px;
  }
}

#header ul.nav {
  @include horizontal-list;
  float: right;
}

#footer ul.nav {
  @include horizontal-list;
  margin-top: 1em;
}
```

Just as the name suggests, Sass mixins *mix in* rules into other rules. We've extracted the rules for the horizontal list into an aptly named mixin using the `@mixin` directive. We then *include* those rules into other rules using the `@include` directive. We no longer need the `.horizontal-list` class since those rules are now mixed into our `ul.nav` rules in our resulting CSS:

```
#header ul.nav {
  float: right;
}

#header ul.nav li {
  float: left;
  margin-right: 10px;
}

#footer ul.nav {
  margin-top: 1em;
}

#footer ul.nav li {
  float: left;
  margin-right: 10px;
}
```

As handy as this is, the real power of Sass mixins comes from combining them with variables to make reusable, parameter-driven blocks of styles. For example, let's suppose we wanted to vary the item spacing in our horizontal list. Find the next code example, 1.1.2.2.mixins-parameters.scss and consider the following changes:

```
@mixin horizontal-list($spacing: 10px) {
  li {
    float: left;
    margin-right: $spacing;
  }
}

#header ul.nav {
  @include horizontal-list;
  float: right;
}

#footer ul.nav {
  @include horizontal-list(20px);
  margin-top: 1em;
}
```

We've updated the mixin and added a `$spacing` parameter with a default value of `10px`. Parameters are no different than the variables we looked at earlier. In this case we've specified a default value so that in the case of the navigation list in the header, we get the default spacing. In the footer, however, we now can pass in a value of `20px` to get a bit more spacing between the list elements, as you can see in the CSS output:

```
#header ul.nav {
  float: right;
}

#header ul.nav li {
  float: left;
  margin-right: 10px;
}

#footer ul.nav {
  margin-top: 1em;
}

#footer ul.nav li {
  float: left;
  margin-right: 20px;
}
```

Sass mixins save you a lot of time, letting you reuse chunks of properties, but

the astute reader might notice that what we've gained in productivity, we may have given back in stylesheet weight since mixin styles are duplicated in each instance they're included. Fear not, with Sass, you've always got options. Selector inheritance deals with just this issue.

1.2.4 Avoid property duplication with selector inheritance

As we've seen, Sass mixins can be a powerful way to avoid duplication when writing your stylesheets. However, since rules are mixed into other classes in the compiled CSS, you're not avoiding duplication entirely. Since CSS file size is important, Sass includes another slightly more complex way of avoiding duplication altogether. Selector inheritance instructs a selector to *inherit* all the styles of another without duplicating the CSS properties. Take for instance the styles for a set of form error messages.

```
.error {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion {
  font-size: 1.2em;
  font-weight: bold;
}

.badError {
  @extend .error;
  border-width: 3px;
}
```

Using selector inheritance, we can instruct `.badError` to inherit from the base `.error` class, yielding:

```
.error, .badError {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion,
.badError.intrusion {
  font-size: 1.2em;
  font-weight: bold;
}

.badError {
  border-width: 3px;
}
```

With a little planning, selector inheritance is a nice way to keep your Sass DRY

and your CSS lean.

1.3 What is Compass?

Like Samneric, the twins in *Lord of the Flies* who lost their individuality and were always referred to by their conjoined names, you might have first been exposed to Sass because a developer friend excitedly told you to check out *Compass-n-Sass*. For newcomers, the line between the two projects is often blurred, much like the distinction between Ruby & Rails, Django & Python, or .NET and C# for those learning a pair of projects at the same time. So, now that we've seen Sass, what is Compass and how does it relate to Sass?

1.3.1 Simple stylesheet projects

Both Sass and Compass are written in Ruby and have their origins in the Ruby on Rails community. But it seemed unfair to all the non-ruby web developers, so Compass provides the necessary tools to help Sass escape it's ruby roots. Whether your need is to simply build an HTML mockup or to integrate Sass into a large application framework like Django, Drupal or .NET, Compass provides a set of tools and configuration options to make wrangling your stylesheets outside of a Ruby-based project a snap.

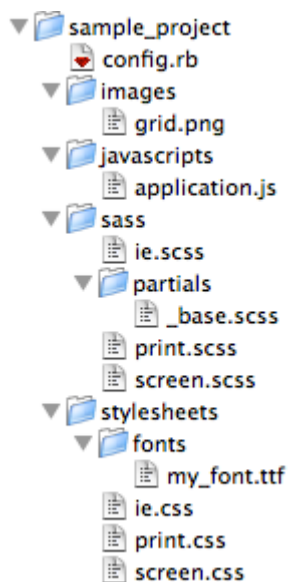


Figure 1.1 A
Stand-Alone Compass
Project

Compass understands that you aren't building stylesheets. You are building a design. As such, Compass wants to know where you keep things like image, font, and javascript files so that it can simplify the management of and references to

those files from within your stylesheets. For example, Compass will help you construct sprite maps and refer to those within your stylesheets; Compass will warn you if you reference an image that doesn't exist via the `image-url()` helper; and compass can embed an image or font into your css so that the browser doesn't have to make another round trip to get that asset.

1.3.2 *jQuery for Stylesheets*

If you've been in web development for a while you might remember the dark ages before Javascript frameworks. It was truly a terrible world -- the smallest quirk in the DOM might send you on a bug hunt for hours. These days there are a half dozen quality free, open source Javascript libraries that you can use to isolate you from the browsers' inconsistent DOM implementations. Thanks to the hard work of the web development community at large, developing with Javascript is actually quite enjoyable these days.

Through the power of Sass, Compass provides you a library of stylesheets that insulate you from cross browser quirks and spare you from the need to remember the specifics needed to perform common design tasks. These tasks include resets, grid layouts, table helpers, CSS3 helpers, sprites, and more. These are tasks you could tackle yourself, and sometimes will, but Compass bundles proven solutions from the design community, letting you focus on getting more done in less time.

The Compass Core stylesheet framework isn't going to make your website pretty. In fact, every feature in the core framework is *design agnostic* so that they can be used with any website design. Website design aesthetics, like all fashions, come and go. So the task of providing well designed website features is left to the Compass community of front end developers and designers through the use of plugins.

1.3.3 *Community Ecosystem*

They say "it takes a village" to raise a child, well it takes a community to manage the complexities of Cross browser development. Through the power of the Open Source development model, compass has a robust foundation for all kinds of css-based frameworks and projects to be distributed, but the real power of Compass rests on and within the Community itself.

In times past, clever designers would blog about a useful design approach and other developers would then copy their code samples and tweak it. But this strategy means that you're left owning the code that you copied and there's no way for the original developer to fix bugs and provide additional enhancements over time.

With compass, stylesheet libraries can be distributed like other software which means fixing a bug or getting support for the latest browsers may just be a simple matter of upgrading and recompiling your stylesheets.

Many community members have taken the initiative to extract useful tools from their own work and distribute them as compass plugins. From layout frameworks to fancy buttons, compass plugins provide get you past the drudgery of building the basics so you can focus on what is unique and special about your website. See Chapter 12 for details about where to find compass plugins and how to install them.

As you progress from Sass novice to Sass assassin, if you're grateful for all the time Sass, Compass, and the community save you, you will be able "pay it forward" by sharing your hard work with others. See chapters 13 and 14 for details on how to build and distribute your stylesheets. It's not as hard as you might think!

1.4 Create a Compass project

If you haven't installed Compass already, go ahead and jump to Appendix A and follow the instructions outlined there. Once you have the bits installed, we'll be ready to start using Compass. Our first task will be creating a Compass project.

Like any good Command Line Interface (CLI), Compass provides substantial help messages for its many options. Let's check your Compass install. Open a terminal window in the root of a new stylesheet project. Now, let's run `compass help`. If you're greeted with help text and command line options, you're good to go. If not, circle back to Appendix A one more time and we'll see you on the flip side.

Let's start by creating a new Compass *project* which is a configuration file and folders for our Sass source and CSS output. We'll call it *sample*.

```
compass create sample
```

Do we need hedgehog here annotating the parts of the CLI command and args?

Now let's list the contents of our new folder:

```
total 8
drwxr-xr-x  6 wynn  staff  204 Jan  3 12:11 .
drwxr-xr-x  3 wynn  staff  102 Jan  3 12:12 ..
drwxr-xr-x  4 wynn  staff  136 Jan  3 12:11 .sass-cache
-rw-r--r--  1 wynn  staff  315 Jan  3 12:11 config.rb
drwxr-xr-x  5 wynn  staff  170 Jan  3 12:11 src
drwxr-xr-x  5 wynn  staff  170 Jan  3 12:11 stylesheets
```

Using the defaults, Compass has unfurled a `config.rb` configuration file, a `src` folder for our Sass source and a `stylesheets` folder for our CSS output. For a full list of Compass configuration options, please consult Appendix B. For now, we'll work with the default settings and set out to tackle some real world CSS problems using Compass.

1.5 Solve real world CSS problems with Compass

Now that we've seen how to create a skeleton Compass project, let's take a look at how Compass can help solve some stylesheet challenges you probably face every day. In the next few sections we'll apply Compass' built-in modules (which are really only nice bundles of Sass mixins and other features) to CSS resets, grid layouts, table formatting, and CSS3 features.

1.5.1 Clear the canvas with resets

Made popular by Eric Meyer and other standards advocates, adding a CSS reset has become the first task for designers when creating a stylesheet. If you've ever used a CSS grid framework, you've used a CSS reset, perhaps without even knowing it. A CSS reset simply removes all intrinsic browser styling from all elements, providing a common blank canvas to add back the styling you want.

Eric's classic reset looks like this:

```
/* v1.0 | 20080212 */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-size: 100%;
  vertical-align: baseline;
  background: transparent;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
blockquote, q {
```

```

    quotes: none;
  }
  blockquote:before, blockquote:after,
  q:before, q:after {
    content: '';
    content: none;
  }

  /* remember to define focus styles! */
  :focus {
    outline: 0;
  }

  /* remember to highlight inserts somehow! */
  ins {
    text-decoration: none;
  }
  del {
    text-decoration: line-through;
  }

  /* tables still need 'cellspacing="0"' in the markup */
  table {
    border-collapse: collapse;
    border-spacing: 0;
  }

```

You might have noticed from the default Sass file in example Section 1.4 that Compass ships with its own reset based on Eric's, allowing you to put all browsers on equal footing with a single line in your Sass file:

```
@import "compass/reset"
```

There's a lot going on in this one line, so let's break it down. We're using the Sass `@import` rule to import the Compass Reset module. A module is simply a standalone portion of the Compass framework that can be added independently to your project. With this one line, the contents of our CSS output file include our CSS reset:

```

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, font, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;

```

```

font-weight: inherit;
font-style: inherit;
font-size: 100%;
font-family: inherit;
vertical-align: baseline;
}

body {
  line-height: 1;
  color: black;
  background: white;
}

ol, ul {
  list-style: none;
}

table {
  border-collapse: separate;
  border-spacing: 0;
  vertical-align: middle;
}

caption, th, td {
  text-align: left;
  font-weight: normal;
  vertical-align: middle;
}

q, blockquote {
  quotes: " " " ";
}

q:before, q:after, blockquote:before, blockquote:after {
  content: " ";
}

a img {
  border: none;
}

```

It should be noted that adding styles to your stylesheet isn't the default behavior of most Compass modules, but since the usual use case is to apply the CSS reset, the Compass Reset module goes ahead and applies the `global-reset` mixin upon import. Let's take a look at that mixin:

```

@mixin global-reset {
  html, body, div, span, applet, object, iframe,
  h1, h2, h3, h4, h5, h6, p, blockquote, pre,
  a, abbr, acronym, address, big, cite, code,
  del, dfn, em, font, img, ins, kbd, q, s, samp,
  small, strike, strong, sub, sup, tt, var,
  dl, dt, dd, ol, ul, li,
  fieldset, form, label, legend,
  table, caption, tbody, tfoot, thead, tr, th, td {

```

```

    @include reset-box-model;
    @include reset-font; }
body {
    @include reset-body; }
ol, ul {
    @include reset-list-style; }
table {
    @include reset-table; }
caption, th, td {
    @include reset-table-cell; }
q, blockquote {
    @include reset-quotation; }
a img {
    @include reset-image-anchor-border; } }

```

Note that Compass is using the Sass `@mixin` and `@include` features we looked at earlier to build the reset. In addition to the `global-reset`, the Reset module includes a number of more surgical reset mixins, including one for HTML5 elements. By simply adding `@include reset-html5` to our Sass file, we get an additional CSS rule in our output for all the HTML5 elements that need some basic styling:

```

article, aside, canvas, details, figcaption, figure, footer, header, hgroup, menu, n
margin: 0;
padding: 0;
border: 0;
outline: 0;
display: block;
}

```

For additional Compass Reset module mixins, be sure and check out the Compass online docs. Now that we've got a handle on resets, let's look at how Compass can help you more effectively use CSS grid frameworks.

1.5.2 Create layouts without a calculator

One of the major trends in CSS of the last couple of years has been the emergence of popular CSS grid frameworks such as Blueprint and 960 Grid System. Grid layouts which have long been a cornerstone of good print design have made their way online as the medium has matured. Grid frameworks allow you to allot a certain number of columns for your layout and then apply a column-based layout with uniform gutters to your content.

Twitter
ENHANSHTH

Download - Templates: Acom, Fireworks, Flash, InDesign, GIMP, Inkscape, Illustrator, OmniGraffle, Photoshop, Visio, Exp Design. Also: PDF sketch sheets + CSS files. Repository at [GitHub](#).

960
GRID SYSTEM

ADS BY FUSION
If you are aren't using it, you're wasting time.

te
textexpander

CUSTOM CSS GENERATOR HTML LAYOUT GENERATOR GRID OVERLAY BOOKMARK

Essence
The 960 Grid System is an effort to streamline web development workflow by providing commonly used dimensions, based on a width of 960 pixels. There are two variants: 12 and 16 columns, which can be used separately or in tandem. [Read more](#).

Dimensions
The 12-column grid is divided into portions that are 60 pixels wide. The 16-column grid consists of 40 pixel increments. Each column has 10 pixels of margin on the left and right, which create 20 pixel wide gutters between columns. [View demo](#).

Purpose
The premise of the system is ideally suited to rapid prototyping, but it would work equally well when integrated into a production environment. There are printable sketch sheets, design layouts, and a CSS file that have identical measurements.

More Columns
For those more comfortable designing on a 24-column grid, an alternative version is also included. It consists of columns 30 pixels wide, with 10 pixel gutters, and a 5 pixel buffer on each side of the container. This keeps text from touching browser chrome — helpful for devices like the iPhone, where a lower-case "i" or "l" might be easily missed. [View demo](#).

Source Order
By utilizing the *push_XX* and *pull_XX* classes, elements can be rearranged, independent of the order in which they appear in the markup. This allows you to keep more pertinent info higher in the HTML, without sacrificing precision in your page layout. For instance, view the source code of this page to see how the *H1* tag has been re-positioned.

Keynote Wireframe Toolkit — 12 col SHOW GRID Formalize CSS — 12 col SHOW GRID

Wireframe Toolkit For Keynote VERSION 2.2 Examples Buzz Media FAQ Tips \$12 BUY NOW

Break the cycle of inconsistent form defaults, style forms with integrity! Free to try, pay me on [Twitter](#)

formalize
TEACH YOUR FORMS SOME BANNERS

Check out our WE'RE HIRING >

Figure 1.2 960.gs - The 960 Grid System CSS Framework

Basically, grid frameworks reduce the math needed to create a nice column layout. They do this with CSS rules that set the layout and widths for a container element as well as each for each possible column width in the grid. Let's look at a snippet from Blueprint.

```
.container {
  width: 950px;
  margin: 0 auto;
}

/* Sets up basic grid floating and margin. */
.column, .span-1, .span-2, .span-3, .span-4, .span-5, .span-6, .span-7, .span-8, .span-9, .span-10, .span-11, .span-12, .span-13, .span-14, .span-15, .span-16, .span-17, .span-18, .span-19, .span-20, .span-21, .span-22, .span-23, .span-24 {
  float: left;
  margin-right: 10px;
}

/* The last column in a row needs this class. */
.last { margin-right: 0; }

/* Use these classes to set the width of a column. */
.span-1 {width: 30px;}
```

```

.span-2 {width: 70px;}
.span-3 {width: 110px;}
.span-4 {width: 150px;}
.span-5 {width: 190px;}
.span-6 {width: 230px;}
.span-7 {width: 270px;}
.span-8 {width: 310px;}
.span-9 {width: 350px;}
.span-10 {width: 390px;}
.span-11 {width: 430px;}
.span-12 {width: 470px;}
.span-13 {width: 510px;}
.span-14 {width: 550px;}
.span-15 {width: 590px;}
.span-16 {width: 630px;}
.span-17 {width: 670px;}
.span-18 {width: 710px;}
.span-19 {width: 750px;}
.span-20 {width: 790px;}
.span-21 {width: 830px;}
.span-22 {width: 870px;}
.span-23 {width: 910px;}
.span-24 {width:950px; margin-right:0;}

```

With these CSS rules in place, we can create a layout content in sixteen column layout simply by adding the `container` class to a container element and a `span-xx` class to each element we want to place on the grid. Laying out content in this way not only lets us prototype more quickly by not having to remember the multiples of 40 between 30 and 950.

So how does Compass improve upon CSS grid frameworks? First, Compass provides support for Grid framework styles as mixins, allowing you to pull in just the features you want to use while avoid littering your HTML markup with extra classes. The second, and perhaps most important, way Compass supports grid frameworks is in the way it changes the way you create frameworks like these, as we'll see later on in [chapter/section??].

Let's create a Compass project using Blueprint. Run the following in a terminal window:

```
compass create my_grid --using blueprint
```

Just as in section 1.4, you should find a freshly stamped Compass project in a folder called `my_grid`, only this time the `screen.scss` has a bit more content. The file is well annotated and provides a quick survey of the most of the Blueprint modules at your disposal along with a set of styles for a basic layout. The first thing to notice here is that column layouts can be *mixed in* to a set of styles. So

instead of setting a class of `span-8` in your HTML, you simply use the `column` Sass mixin:

```
@include column($sidebar-columns);
```

Also note the variable `$sidebar-columns`. This is extremely powerful because now, thanks to Sass, we can make our layouts variable-driven. We can rapidly prototype and play with different layouts including numbers of columns, gutter width, and sidebar sizes all by changing a few variables at the top of our Sass file. To do this in traditional CSS grid frameworks, you'd have to do the math to create those CSS layouts, and then change the CSS classes in your markup as well.

We won't go into all the aspects of the Blueprint grid here. We'll jump into using Blueprint with Compass a bit later in Chapter 6. We're going to continue our survey of real-world Compass application by taking a look at the Compass table helper.

1.5.3 Zebra-stripe like a pro with table helpers

Continuing our overview of Compass features, let's look at the Compass table helpers, a set of Sass mixins that make prettifying your HTML tables a bit easier. Let's look at example:

```
@import "compass/reset"
@import "compass/utilities/tables";

table {
  $table-color: #666;
  @include table-scaffolding;
  @include inner-table-borders(1px, darken($table-color, 40%));
  @include outer-table-borders(2px);
  @include alternating-rows-and-columns($table-color, adjust-hue($table-color, -120d
```

Now let's break this down. We import the table helpers using the `@import` rule. This provides four mixins for our use. The `table-scaffolding` provides base styles for our `th` and `td` elements that we stripped with our CSS reset as well as a often-used pattern of right alignment for numeric columns. Here's the source for this mixin:

```
@mixin table-scaffolding {
  th {
    text-align: center;
    font-weight: bold; }
  td,
```

```
th {
  padding: 2px;
  &.numeric {
    text-align: right; } } }
```

The `inner-table-borders` and `outer-table-borders` mixins work as advertised adding borders to the table and to cells within the table.

Lastly, the `alternating-rows-and-columns` mixin is an easy way to add zebra-stripping to your HTML table. You might ask why we wouldn't use the `:nth-child`, `:even`, or `:odd` CSS pseudo selectors for this task, and you'd be right. That's exactly what Compass is doing under the hood. However, this mixin provides some additional support for class name based striping as well as color intersections. Let's look at the source:

```
@mixin alternating-rows-and-columns($seven-row-color, $odd-row-color, $dark-intersect
th {
  background-color: $header-color;
  &.even, &:nth-child(2n) {
    background-color: $header-color - $dark-intersection; } }
tr.odd {
  td {
    background-color: $odd-row-color;
    &.even, &:nth-child(2n) {
      background-color: $odd-row-color - $dark-intersection; } } }
tr.even {
  td {
    background-color: $seven-row-color;
    &.even, &:nth-child(2n) {
      background-color: $seven-row-color - $dark-intersection; } } }
tfoot {
  th, td {
    background-color: $footer-color;
    &.even, &:nth-child(2n) {
      background-color: $footer-color - $dark-intersection; } } } }
```

Note the color values are not only variables, they're employing a bit of math to ensure proper contrast for readability. We'll learn more about how Sass deals with variables and math in the next Chapter. Let's keep moving with a look at how Compass means never having to write vendor prefixes again.

1.5.4 Easy CSS3 without vendor prefixes

When CSS3 started gaining adoption by modern browsers, designers were excited to start using CSS for tasks that used to require stupid stylesheet tricks. We were so excited that we could now make those glorious rounded corners with a few lines of CSS we didn't mind much the vendor prefixes that came with them. Vendor prefixes are those `-webkit` and `-moz` bits that browsers add on to CSS features that have experimental support. In its simplest form, it means that to give a `<div>` a set of rounded corners with a `5px` border radius, we have to resort to CSS like:

```
.rounded {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}
```

As usual, Compass can save us from the repetition with a set of border radius mixins found in the Compass CSS3 module. Let's import the module into our Sass file and include the mixin:

```
@import "compass/css3";
.rounded {
  @include border-radius(5px);
}
```

This will yield the following CSS:

```
.rounded {
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  -o-border-radius: 5px;
  -ms-border-radius: 5px;
  -khtml-border-radius: 5px;
  border-radius: 5px;
}
```

Not only did we save our fingers from the common repetition of `-webkit` and `-moz` but we're also being good designers and supporting the other common vendor namespaces as well. While that bit of repetition isn't horrible, what if we only want one of the four corners to be rounded? Well, the Mozilla folks don't yet see eye-to-eye with the rest of the field on the best way to make that happen so we're left with:

```
.rounded-one {
  -moz-border-radius-topleft: 5px;
```

```
-webkit-border-top-left-radius: 5px;
}
```

That's where Compass shines. We can target a single corner for a border radius with the `border-corner-radius` mixin.

```
.rounded-one {
  @include border-corner-radius(top, left, 5px);
}
```

This will give us the CSS we want, Mozilla quirk included.

```
.rounded-one {
  -moz-border-radius-topleft: 5px;
  -webkit-border-top-left-radius: 5px;
  -o-border-top-left-radius: 5px;
  -ms-border-top-left-radius: 5px;
  -khtml-border-top-left-radius: 5px;
  border-top-left-radius: 5px;
}
```

That's just the tip of the tip of the tip of the CSS3 iceberg in Compass. We'll take a deeper look at all the timesaving features in Chapter 9.

1.6 Summary

In this first chapter we've looked at the case for CSS preprocessing. We've also taken a quick look at four key features of Sass: variables, nested selectors, mixins, and selector inheritance. We also took a look at some real world applications of Sass included in the Compass framework including CSS resets, grids, table styling, and CSS3 rounded corners.

In the next chapter we'll dive a bit deeper into Sass syntax including color functions and scripting support. Once we get a bit more Sass under our belts, we can take a deeper look at Compass in the following chapters.