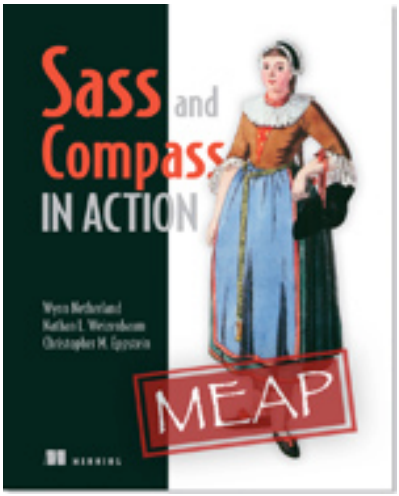
MANNING PUBLICATIONS



Sass and Compass in Action
By Wynn Netherland, Nathan Weizenbaum, and Chris Eppstein

The complexity of the web is not just limited to HTML markup. Due to their heavy reliance on external resources like images, other CSS files and fonts, stylesheets can become a serious maintenance burden. This article, based on chapter 7 of Sass and Compass in Action, explains how to use Compass helpers and configuration to generate URLs to your assets in a consistent way that lets you write the code once and then update your configuration as your asset serving needs change during the transition from prototype to production.

Abstracting URLs

Where are your images? That might sound like a dumb question; your images are right there in your project folder! Sure, they're easy enough to find, but you're a human. If you were a web browser, you'd have a harder time tracking them down. Your images start their long and arduous journey in your project's images directory, but they may soon find themselves packaged, copied, deployed, unpackaged, URL-re-written, compressed, cached, and finally served from one or more places on the Internet.

It's not uncommon for a project to change where and how it stores and references its images three or more times. When you use Compass, you will find it easier to generate and change where and how you store your images, but the benefits don't stop there. While Compass makes URLs for you, it also makes sure the image really exists and that stale images don't get stuck in the browser cache.

Generating URLs to assets

CSS provides the `url` function to denote URLs:

```
background-image: url('/logo.png');
```

URLs identify a resource anywhere on the Internet, but when we're referring to our own assets, we often use relative URLs and the browser resolves the missing pieces of information based on what it knows about the current request. Before we go on though, let's review some terminology related to URLs.

There are four kinds of URLs that can be specified in CSS depending on which parts of the fully qualified URL are omitted.

Table 1 The Four Types of URLs

Example	Type	Description
`url('http://www.example.com/logo.png')`	Absolute URL	The details of the originating request do not matter in this case.

`url('logo.png')`	Relative URL	The browser resolves this URL relative to the request that served it, which in this case is the CSS stylesheet, not the web page. So, if the stylesheet was at http://www.example.com/stylesheets/application.css, this URL points to http://www.example.com/stylesheets/logo.png.
`url('/logo.png')`	Root relative URL	The browser resolves the URL against the protocol and domain of the CSS stylesheet. So, if the stylesheet was http://www.example.com/stylesheets/application.css, this URL points to http://www.example.com/logo.png.
`url('//imgs.example.com/logo.png')`	Protocol relative URL	The browser resolves the URL using the domain specified, but with the same protocol as the originating request for the CSS stylesheet. This type of URL is especially useful when serving assets from a different domain than your main website. So if the stylesheet was https://www.example.com/stylesheets/application.css this URL points to https://imgs.example.com/logo.png

While you can still use any of these URL types in Sass, Compass best practices dictate that you use asset helper functions to refer to your own assets.[1] Compass provides three asset helpers, but to each of them you always pass a path that is relative to that asset class's directory—never relative to the stylesheet:

- `image-url('logo.png')`—References the file `logo.png` saved at the root of your images directory.
- `font-url('arial.ttf')`—References the file `arial.ttf` saved at the root of your fonts directory.
- `stylesheet-url('randomfile.xml')`—References the file `randomfile.xml` saved at the root of your CSS directory.

> **NOTE** You might have noticed that there is no URL helper for JavaScript. The JavaScript configuration option exists so that Compass extensions can provide JavaScript files as part of their installation. Similarly there is no URL generator for the Sass directory because that is merely an aspect of development; the stylesheet_url will point to the location where your CSS files live.

The reason that Compass has chosen this approach is that it makes imports and refactoring much simpler. As you will soon see, it is possible to generate all four kinds of the URLs that CSS supports using this single syntax.

Your project configuration tells Compass where to find your assets so that you can stop caring about how to refer to them in your stylesheets and leave that as an aspect of configuration that will change over time as your stylesheets move from prototype to production to scaling your website or application.

But wait, there's more! For a limited time, Compass will check to make sure your URLs are valid and up to date during compilation!

Avoiding broken links

You're human; sometimes you make mistakes: maybe a typo, maybe an image gets renamed but you miss a reference to it. It happens, don't beat yourself up over it. When you refer to an image using the `image-url($path)` helper function, Compass will verify that the file exists. If it doesn't, it will print out a warning to your console during compilation. Similarly, missing fonts will be noticed when you use the `font-url($path)` helper.

[1] URL Helpers: http://compass-style.org/reference/compass/helpers/urls/

Observe the following output from a compass compilation run where an image is not found. If this were in your console, the WARNING line would be colored red:

```
[~/Projects/my_compass_project] compass compile
directory stylesheets/
   create stylesheets/ie.css
WARNING: 'missing.png' was not found (or cannot be read) in images/
   create stylesheets/main.css
   create stylesheets/print.css
```

Of course, Compass can't actually ensure that you won't have broken links. A configuration error or a change that occurs after compilation will still break your site, but this simple check can save many hours of painful debugging.

Have you ever spent 10 minutes trying to figure out why an image isn't displaying correctly only to discover that the browser was caching it? So have we, but not since starting to use Compass to generate our URLs. Read on to see how.

Avoiding stale images with cache busting

Another common problem both during development and across production deployments is that browsers are lazy; they simply don't like to download things.

I'm sure it's very tedious for them. So they cache your images and other assets in case they need them later, and they often do. This is great for users; it makes their browsing experience much nicer. But it's a pain for us web developers. If you change an image, those users who have recently downloaded it won't notice. And it's really a shame, because the new image is clearly much better. To work around this, compass adds a query parameter to then end of each asset based on its modification time. Your web server will still serve it just fine, but when the query parameter changes, it will force the browser to ask for the image again.

For example:

```
#logo { background-image: image-url('logo.png'); }
```

Might be compiled to:

```
#logo { background-image: url('/images/logo.png?1298578273'); }
```

It is also possible for you to configure what cache buster parameter gets created if timestamps are not a good approach for your needs. For example, you could simply increment a deployment count before each deployment or you could use your source control's revision number for that file. Doing this requires that you write a little Ruby code. For example:

```
# Increment the deploy_version before every
# release to force cache busting.
asset_cache_buster do |http_path, real_path|
  "v=1"
end
```

would cause Compass to now generate the following output for our Logo image:

```
#logo { background-image: url('/images/logo.png?v=1'); }
```

Using a query parameter as a cache busting strategy might interfere with the ability for some proxies to cache your assets. (The query string makes them scared that the asset might be dynamically generated). If this is a problem for you, it's possible to disable the cache buster by adding the following line to your Compass configuration:

```
asset_cache_buster :none
```

However, if you want to maximize cache-ability of your assets while also busting the cache, the best way to bust the cache by rewriting the path to the asset instead. With some corresponding webserver configuration, we can generate a URL more like this:

```
#logo { background-image: url('/images/logo-1307943914.png'); }
```

You'll need to configure your webserver so that it knows how to map the timestamped path to the real path. How to do this is specific to your webserver, but the Compass configuration looks something like this:

Listing 1 Defining a path-based asset cache buster

```
asset_cache_buster do |path, real_path|
  if File.exists?(real_path)                          #1
    pathname = Pathname.new(path)                     #2
    modified_time = File.mtime(real_path)             #3
    new_path = "%s/%s-%s%s" % [                       #4
       pathname.dirname,
       pathname.basename(pathname.extname),
       modified_time.strftime("%s"),
       pathname.extname
    ]
    {:path =&gt; new_path}                            #5
  end
end
```

#1 Always check if the file really exists.
#2 Path names are easier to work with than strings.
#3 The last modified time
#4 Construct a new path from 4 strings
#5 This is a special return format for path-based assets

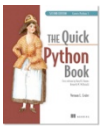
As you can see, arbitrarily complex logic can be used within the Compass configuration file because it is actually just a Ruby script. For example, some users integrate with their Content Distribution Network (CDN) or generate MD-5 fingerprints for each asset. However, more complex logic is outside of the scope of this article, so please contact the Compass mailing list[2] or your nearest Rubyist if you need help crafting custom code.

Summary

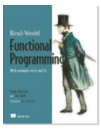
We explored the Compass best practices for generating URLs. Compass makes URLs for you, but it also makes sure the image really exists and that stale images don't get stuck in the browser cache.

[2] Mailing List: http://groups.google.com/group/compass-users

Here are some other Manning titles you might be interested in:



The Quick Python Book, Second Edition
Vernon L. Ceder



Real-World Functional Programming
Tomas Petricek with Jon Skeet

Hadoop in Action
Chuck Lam

Last updated: November 10, 2012