pyScss Documentation

Release 1.3.0.dev1

German M. Bravo (Kronuz)

June 09, 2016

Contents

pyScss is a Python implementation of Sass, a CSS preprocessing language that adds variables, expressions, nested rules, mixins, inheritance, and other features that help ease the maintenance of large stylesheets.

pyScss also includes support for Compass, and has an extension mechanism for adding your own custom behavior.

pyScss is not yet fully compatible with the canonical Ruby implementation, but we're getting there and constantly improving. Please feel free to file a GitHub issue for anything pyScss gets wrong. Contents:

CHAPTER 1

Installation and usage

1.1 Installation

pyScss requires only Python 2.6 or later, including Python 3.x. PyPy is also known to work. Install with pip:

```
pip install pyScss
```

It has a handful of pure-Python dependencies, which pip should install for you:

- `six`
- `enum34` (for Python 3.3 and below)
- `pathlib` (for Python 3.3 and below)

There's also an optional C speedup module, which requires having `libpcre` and its development headers installed, with UTF-8 support enabled (which it is by default).

1.2 Usage

Run from the command line by using `-m`:

```
python -m scss < file.scss
```

Specify directories to search for imports with `-I`. See `python -mscss --help` for more options.

Note: `-mscss` will only work in Python 2.7 and above. In Python 2.6, `-m` doesn't work with packages, and you need to invoke this instead:

```
python -m scss.tool
```

1.3 Interactive mode

To get a REPL:

```
python -mscss --interactive
```

Example session:

```
$ python scss.py --interactive
>>> @import "compass/css3"
>>> show()
['functions', 'mixins', 'options', 'vars']
>>> show(mixins)
['apply-origin',
    'apply-transform',
    ...
    'transparent']
>>> show(mixins, transparent)
@mixin transparent() {
    @include opacity(0);
}
>>> 1px + 5px
6px
>>> _
```

1.4 Compass example

With `--load-path` set to Compass and Blueprint roots, you can compile with Compass like with the following:

```
@option compress: no;

$blueprint-grid-columns : 24;
$blueprint-grid-width   : 30px;
$blueprint-grid-margin  : 10px;
$font-color             : #333;

@import "compass/reset";
@import "compass/utilities";
@import "blueprint";

// your code...
```

CHAPTER 2

Python API

2.1 Legacy API

Warning: This API is still supported while the new API below is worked out, but it's slated for deprecation and eventual removal. If you don't need any of the features not yet available with the new API, consider porting as soon as possible.

2.1.1 Compiling files

Very basic usage is simple enough:

```
from scss import Scss
css = Scss()
css.compile("a { color: red + green; }")
```

2.1.2 Configuration

There are several configuration variables in the `scss.config` module that you may wish to change.

`PROJECT_ROOT`: Root of your entire project. Used only to construct defaults for other variables. Defaults to the root of the pyScss installation, which is probably not what you want.

`LOAD_PATHS`: An iterable of paths to search when using``@import``.

`STATIC_ROOT`: Used for finding sprite files. Defaults to `$PROJECT_ROOT/static`.

`ASSETS_ROOT`: Generated sprites are saved here. Defaults to `$STATIC_ROOT/assets`.

`CACHE_ROOT`: Used for storing cached sprite information. Defaults to `ASSETS_ROOT`.

`STATIC_URL`: URL equivalent to `STATIC_ROOT`. Defaults to `static/`.

`ASSETS_URL`: URL equivalent to `ASSETS_ROOT`. Defaults to `static/assets/`.

`SPRTE_MAP_DIRECTION`: Direction in which to arrange sprites in a spritesheet. Defaults to `vertical`; may be changed to `horizontal`, `diagonal`, or `smart`.

`VERBOSITY`: Increase spew from the compiler. Defaults to `1`.

`DEBUG`: Set to true to make parse errors fatal. Defaults to false.

2.1.3 Django example

A rough example of using pyScss with Django:

```
import os
import fnmatch

import scss

from django.conf import settings
from django.utils.datastructures import SortedDict
from django.contrib.staticfiles import finders


def finder(glob):
    """
    Finds all files in the django finders for a given glob,
    returns the file path, if available, and the django storage object.
    storage objects must implement the File storage API:
    https://docs.djangoproject.com/en/dev/ref/files/storage/
    """
    for finder in finders.get_finders():
        for path, storage in finder.list([]):
            if fnmatch.fnmatchcase(path, glob):
                yield path, storage


# STATIC_ROOT is where pyScss looks for images and static data.
# STATIC_ROOT can be either a fully qualified path name or a "finder"
# iterable function that receives a filename or glob and returns a tuple
# of the file found and its file storage object for each matching file.
# (https://docs.djangoproject.com/en/dev/ref/files/storage/)
scss.config.STATIC_ROOT = finder
scss.config.STATIC_URL = settings.STATIC_URL

# ASSETS_ROOT is where the pyScss outputs the generated files such as spritemaps
# and compile cache:
scss.config.ASSETS_ROOT = os.path.join(settings.MEDIA_ROOT, 'assets/')
scss.config.ASSETS_URL = settings.MEDIA_URL + 'assets/'

# These are the paths pyScss will look ".scss" files on. This can be the path to
# the compass framework or blueprint or compass-recepies, etc.
scss.config.LOAD_PATHS = [
    '/usr/local/www/sass/frameworks/',
    '/Library/Ruby/Gems/1.8/gems/compass-0.11.5/frameworks/compass/stylesheets/',
    '/Library/Ruby/Gems/1.8/gems/compass-0.11.5/frameworks/blueprint/stylesheets/',
]

# This creates the Scss object used to compile SCSS code. In this example,
# _scss_vars will hold the context variables:
_scss_vars = {}
_scss = scss.Scss(
    scss_vars=_scss_vars,
    scss_opts={
        'compress': True,
        'debug_info': True,
    }
)
```

```
# 1. Compile from a string:
compiled_css_from_string = _scss.compile('@import "file2"; a {color: red + green; }')

# 2. Compile from a file:
compiled_css_from_file = _scss.compile(scss_file='file1.scss')

# 3. Compile from a set of files (use SortedDict or collections.OrderedDict to
# maintain the compile order):
_scss._scss_files = SortedDict((
    ('file2.scss', open('file2.scss').read()),
    ('file3.scss', open('file3.scss').read()),
    ('file4.scss', open('file4.scss').read()),
))
compiled_css_from_files = _scss.compile()
```

Note: The API here is likely to be improved in 1.3, to avoid the need for calling underscored functions.

2.2 New API

The simplest example:

```
from scss.compiler import compile_string

print(compile_string("a { color: red + green; }"))
```

`scss.compiler.compile_string()` is just a simple wrapper around the `scss.compiler.Compiler` class:

```
from scss.compiler import Compiler

compiler = Compiler()
print(compiler.compile_string("a { color: red + green; }"))
```

The most common arguments passed to *Compiler* are:

search_path A list of paths to search for `@import`s. May be either strings or `pathlib.Path` objects.

2.3 Extending pyScss

A significant advantage to using pyScss is that you can inject Python values and code into the Sass compilation process.

2.3.1 Injecting values

You can define Sass values by creating and populating a `scss.namespace.Namespace`:

```
from scss.namespace import Namespace
from scss.types import String

namespace = Namespace()
namespace.set_variable('$base-url', String('http://localhost/'))
compiler = Compiler(namespace=namespace)
compiler.compile_string('div { background: url($base-url); }')
```

Now, `$base-url` will be available to the compiled Sass code, just like any other variable. Note that the value given must be one of the Sass types defined in `scss.types`.

2.3.2 Injecting functions

You can inject functions the same way:

```
def square(x):
    return x * x

namespace.set_function('square', 1, square)
```

This creates a function `square` for use in your Sass source. Optional arguments, keyword arguments, and slurpy arguments are all supported automatically. The arguments are Sass types, and the return value must be one as well.

The second argument is the arity — the number of required arguments, or None if any number of arguments is allowed. Sass functions can be overloaded by arity, so this is required. For functions with optional arguments, adding the same function multiple times can be tedious and error-prone, so the `declare` decorator is also available:

```
@namespace.declare
def square(x):
    return x * x
```

This will inspect the arguments for you and register the function with all arities it accepts. The function name is determined from the Python name: underscores become hyphens, and trailing underscores are removed. If you'd prefer to be more explicit, there's also a `declare_alias`:

```
@namespace.declare_alias('square')
def square(x):
    return x * x
```

2.4 API reference

2.4.1 scss.compiler

2.4.2 scss.namespace

2.4.3 scss.extension

CHAPTER 3

pyScss syntax

3.1 Supported Sass features

pyScss is mostly compatible with Sass 3.2 and has partial support for the upcoming Sass 3.3. The canonical syntax reference is in the Sass documentation: http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html

3.1.1 Both syntaxes

SCSS (CSS3 superset) is the primary syntax, but there's experimental support for the SASS (YAML-like) syntax.

3.1.2 Built-in functions

All of the Sass 3.2 functions described in the Sass documentation are supported.

3.1.3 Rule nesting

Rule/selector nesting and the `&` parent-reference selector are both supported.

Example:

```
.selector {
    a {
        display: block;
    }
    strong {
        color: blue;
    }
}
```

Produces:

```
.selector a {
    display: block;
}
.selector strong {
    color: blue;
}
```

3.1.4 Variables, data types

Variables are supported. All of the Sass data types—strings, numbers, booleans, colors, lists, maps, and null—are supported.

Example:

```
$main-color: #ce4dd6;
$style: solid;
$side: bottom;
#navbar {
    border-#{$side}: {
        color: $main-color;
        style: $style;
    }
}
```

Produces:

```
#navbar {
    border-bottom-color: #ce4dd6;
    border-bottom-style: solid;
}
```

3.1.5 Functions and mixins

`@function`, `@mixin`, and `@include` (optionally with `@content`) are supported.

Named arguments (`foo($name: value)`) and slurpy arguments (`foo($args...)`) are also supported.

Example:

```
@mixin rounded($side, $radius: 10px) {
    border-#{$side}-radius: $radius;
    -moz-border-radius-#{$side}: $radius;
    -webkit-border-#{$side}-radius: $radius;
}
#navbar li { @include rounded(top); }
#footer { @include rounded(top, 5px); }
#sidebar { @include rounded(left, 8px); }
```

Produces:

```
#navbar li {
    border-top-radius: 10px;
    -moz-border-radius-top: 10px;
    -webkit-border-top-radius: 10px;
}
#footer {
    border-top-radius: 5px;
    -moz-border-radius-top: 5px;
    -webkit-border-top-radius: 5px;
}
#sidebar {
    border-left-radius: 8px;
    -moz-border-radius-left: 8px;
    -webkit-border-left-radius: 8px;
}
```

3.1.6 Rule extension

`@extend` is supported, though some particularly thorny edge cases may not produce output identical to the reference compiler.

Example:

```
.error {
    border: 1px #f00;
    background-color: #fdd;
}
.error.intrusion {
    background-image: url("/image/hacked.png");
}
.seriousError {
    @extend .error;
    border-width: 3px;
}
```

Produces:

```
.error,
.seriousError {
    border: 1px red;
    background-color: #fdd;
}
.error.intrusion,
.seriousError.intrusion {
    background-image: url("/image/hacked.png");
}
.seriousError {
    border-width: 3px;
}
```

3.1.7 Conditions

`@if`, `@else if`, and `@else` are supported.

3.1.8 Loops

Both types of iteration are supported:

```
@for $n from 1 through 9 {
    .span-#{$n} { width: $n * 10%; }
}

@each $color in red, blue, yellow {
    .button-#{$color} {
        background-color: $color;
    }
}
```

Additionally, the unpacking-iteration syntax in Sass trunk is supposed; see *Maps*.

3.1.9 Maps

pyScss has experimental support for maps, a data type recently added to Sass trunk. Maps are defined with colons inside parentheses:

```
$colors: (
    text: black,
    background: white
);
```

Keys may be any Sass expression, not just strings.

Maps are manipulated with a handful of map functions:

```
a {
    color: map-get($colors, text);
    background-color: map-get($colors, background);
}
```

A map is semantically equivalent to a list of 2-lists, stored in the order they appeared when the map was defined. Any list operation will work on a map:

```
div {
    // I don't know why you'd do this  :)
    margin: nth($colors, 1);  // => text, black
}
```

Maps may be iterated over with `@each`, of course, but each item will be a somewhat clumsy 2-list. Instead, you can give multiple variables to do an unpacking iteration:

```
@each $key, $value in $colors {
    // I don't know why you'd do this either!
    [data-style=$key] {
        color: $value;
    }
}
```

This syntax works on any list-of-lists.

3.1.10 Everything is a list

Another change borrowed from Sass trunk: any scalar type (string, number, boolean, etc.) will also act as a list of one element when used where a list is expected. This is most useful when writing Python extensions, but may also save you from checking `type-of` in a complex API.

3.2 Compass support

An arbitrary cross-section of Compass 0.11 is supported:

- **Math functions**: `sin`, `cos`, `tan`, `round`, `ceil`, `floor`, `pi`, `e`
- **Images**: `image-url`, `image-width`, `image-height`...
- **Embedded (inline) images**: `inline-image`

Note: Currently, Compass support is provided by default, which has led to some surprising behavior since parts of Compass conflict with parts of CSS3. In the future, Compass will become an extension like it is for Ruby, and you will have to opt in.

3.2.1 Sprites

Example:

```
$icons: sprite-map("sociable/*.png"); // contains sociable/facebook.png among others.
div {
    background: $icons;
}
@each $icon in sprites($icons) {
    div .#{$icon} {
        width: image-width(sprite-file($icons, $icon));
        height: image-height(sprite-file($icons, $icon));
        background-position: sprite-position($icons, $icon);
    }
}
```

...generates a new sprite file and produces something like:

```
div {
    background: url("/static/assets/u8Y7yEQL0UffAVw5rX7yhw.png?_=1298240989") 0px 0px no-repeat;
}
div .facebook {
    width: 32px;
    height: 32px;
    background-position: 0px 0px;
}
div .twitter {
    width: 32px;
    height: 32px;
    background-position: 0px -32px;
}
...
```

3.3 pyScss-specific extensions

pyScss supports some constructs that upstream Sass does not, for various reasons. Listed here are "blessed" features in no danger of being removed, though you should avoid them if you're at all interested in working with the reference compiler.

There are also some deviations that only exist for backwards compatibility; you should **not** rely on them, they will start spewing warnings at some point in the future, and eventually they will disappear. They are listed separately in *Deprecated features*.

3.3.1 `@option`

Compiler options may be toggled at runtime with `@option`. At the moment the only supported option is `compress`, to control whether the output is compressed:

```
@option compress: true;
```

3.3.2 Multiplying strings by numbers

Much like in Python, this works:

```
content: "foo" * 3;  // => "foofoofoo"
```

This is a runtime error in the reference compiler.

3.4 Deprecated features

3.4.1 Brackets to delimit expressions

In an expression, square brackets are equivalent to parentheses:

```
margin-top: [1px + 2px] * 3;  // => 9px
```

This is a holdover from xCSS and will be removed in the future.

3.4.2 `extends`

There's an alternative syntax for `@extend`:

```
a extends b {
    ...
}
```

This is identical to:

```
a {
    @extend b;
    ...
}
```

This is a holdover from xCSS and will be removed in the future.

3.4.3 `self` selector

`self` is an alias for `&`:

```
a {
    self:hover {
        text-decoration: underline;
    }
}
```

This is a holdover from xCSS and will be removed in the future.

3.4.4 `@variables` block

Variables may be declared in a dedicated block:

```
@variables {
    $color: red;
}
```

`@vars` is an alias for `@variables`.

This is a holdover from xCSS and will be removed in the future.

3.4.5 `+foo` to include a mixin

This:

```
div {
    +border-radius 3px;
}
```

Is equivalent to this:

```
div {
    @include border-radius(3px);
}
```

This is the same as the Sass syntax, but causes some parsing ambiguity, since `+foo` with a block could be either a nested CSS block with a sibling selector or a mixin call. Its future is uncertain, but you should probably avoid using it in SCSS files.

3.4.6 Soft errors

pyScss is much more liberal in what it accepts than the reference compiler; for example, rules at the top level and missing closing braces are accepted without complaint, and attempting to use a non-existent mixin only results in a warning.

pyScss 2.0 is likely to be much stricter; don't rely on any particular abuse of syntax to work in the future.

3.4.7 Operations on lists

Binary operations with a list on the left-hand side are performed element-wise:

```
p {
    margin: (1em 0 3em) * 0.5;  // => 0.5em 0 1.5em
}
```

Given that future versions of the reference compiler are likely to introduce built-in list operations, the future of this feature is unclear.

3.4.8 Mixin "injection"

A mixin defined like this:

```
@mixin foo(...) {
    // ...
}
```

will accept **any** keyword arguments, which will be available as variables within the mixin.

This behavior exists for historical reasons and due to the lack of a `**kwargs` equivalent within Sass. Its usage makes mixin behavior harder to understand and you should not use it.

3.5 Unsupported Sass features

Some Sass features are not supported or have some gaps. Each of these may be considered a bug.

3.5.1 CLI

pyScss's command-line arguments are not entirely compatible with those of the reference compiler.

3.5.2 Sass 3.3

The following Sass 3.3 improvements are not yet implemented, but are planned for the near future:

- Use of `&` in expressions.
- `@at-root`
- Source map support.
- Using `...` multiple times in a function call, or passing a map of arguments with `...`. Likewise, `keywords()` is not implemented.
- `unique-id()`, `call()`, and the various `*-exists()` functions are not implemented.

CHAPTER 4

Back matter

4.1 Reporting bugs

If you have any suggestions, bug reports, or minor annoyances, please report them to the issue tracker on GitHub: http://github.com/Kronuz/pyScss/issues

4.2 Contributing

Please send us pull requests on GitHub! https://github.com/Kronuz/pyScss

4.3 Running the test suite

The test suite is built atop the excellent py.test library, and can be run with:

```
py.test
```

from the root of a source checkout.

Most of the tests are pairs of input/output files in `scss/tests/files`; the test suite scans for these, compiles all the `.scss` files, and compares the output with the `.css` file of the same name. To run only particular tests, you can pass them directly as filenames:

```
py.test scss/tests/files/general/000-smoketest.scss
```

There are also several tests borrowed from the Ruby and C implementations. Many of these don't work (due to missing features, different error messages, slightly different formatting, etc.), so to reduce the useless noise produced by a test run, you must explicitly opt into them with `--include-ruby`, even when using a file filter. These files are in the `from-ruby/` and `from-sassc/` subdirectories.

Other than the borrowed tests, the directory names are arbitrary.

4.4 License and copyright

pyScss is inspired by and partially derived from various projects:

- Compass © 2009 Christopher M. Eppstein
- Sass © 2006-2009 Hampton Catlin and Nathan Weizenbaum
- xCSS © 2010 Anton Pawlik

Special thanks to Yelp for allowing Eevee to contribute to pyScss during working hours. Yelp does not claim copyright.

4.5 Changelog

4.5.1 1.3.5 (June 8, 2016)

- The new `less2scss` module attempts to convert Less syntax to SCSS.
- The `*-exists` functions from Sass 3.3 are now supported.
- The contents of a file `@import`-ed in the middle of an input file now appears in the expected place, not at the end of the output.
- Double-slashes within URLs, as often happens with base64-encoded data URLs, are no longer stripped as comments.
- Nesting selectors that end with a combinator, e.g. `div > { p { ... } }`, now works correctly.
- `invert()` is now left alone when the argument is a number, indicating the CSS filter rather than the Sass function.
- `if()` now evaluates its arguments lazily.
- `str-slice()` now silently corrects out-of-bounds indices.
- `append()` now defaults to returning a space-delimited list, when the given list has fewer than two elements.
- `-moz-calc` and `-webkit-calc` are recognized as variants of the `calc()` CSS function.
- Filenames containing dots can now be imported.
- Properties with a computed value of `null` are now omitted from the output.
- The `opacity` token in IE's strange `alpha(opacity=N)` construct is now recognized case-insensitively.
- The Compass gradient functions now recognize `currentColor` as a color.
- The fonts extension should now work under Python 3.
- Escaped astral plane characters no longer crash narrow Python 2 builds.
- The alpha value in `rgba(...)` is no longer truncated to only two decimal places.
- Some edge cases with float precision were fixed, so 742px - 40px is no longer 701.99999999px.

4.5.2 1.3.4 (Dec 15, 2014)

- The modulus (`%`) operator is now supported.
- `/` and − inside an expression work correctly; this fixes some cases of using vanilla CSS's `/` syntax.
- Relative imports have been more fixed.
- Line numbers in error messages are... more likely to be correct.
- Sass at-blocks now parse correctly even when there's no space after the block name, e.g. `@if(foo){...}`.
- A few more cases of `#{...}` being replaced lexically have been switched to real parsing.

With these changes, pyScss can now successfully compile Zurb Foundation 5.

4.5.3 1.3.3 (Nov 18, 2014)

- URLs with quotes now parse as the *Url* type, not as generic functions. Fixes some uses of `@import`.
- A `return` got lost in the Compass gradient code, which would break some uses of gradients.
- Some API work in an attempt to get django-pyscss working against 1.3.

4.5.4 1.3.2 (Oct 17, 2014)

- Fix another couple embarrassing bugs, this time with the CLI.
- Fix the auto behavior of `join()` to match Ruby.
- Fully allow arbitrary expressions as map keys; previously, this only worked for the first key. LL(1) is hard.
- Restore Python 3.2 compatibility.
- Travis CI and Coveralls are now enabled.

4.5.5 1.3.1 (Oct 16, 2014)

Fixes an embarrassing crash when compiling multiple files together.

4.5.6 1.3.0 (Oct 15, 2014)

This is a somewhat transitional release along the road to 2.0, which will remove a lot of deprecated features.

Sass/CSS compatibility

- Importing files from a parent directory with `../` now works (as long as the imported file is still on the search path).
- Multiple CSS imports on the same line are now left unchanged. (Previously, the line would be repeated once for each import.)
- CSS 3 character set detection is supported.
- CSS escapes within strings are supported (though, unlike Sass, are usually printed literally rather than escaped).
- Map keys may now be arbitrary expressions.
- Slurpy named arguments are now supported.
- `!global` on variable assignments is now supported (and does nothing, as in Sass).
- rebeccapurple is understood as a color name.

Additionally, a great many more constructs should now parse correctly. By default, when pyScss encounters a parse error, it replaces any interpolations and variables, and treats the result as a single opaque string.

This was the only way syntax like `url(http://foo/bar)` was recognized, since a colon is usually not allowed in the middle of a bareword. As a result, code like `background: url(...) scale-color(...);` didn't work, because the url would fail to parse and so pyScss would never even know that `scale-color` is supposed to be a function call.

Now, the parser understands most of the unusual quirks of CSS syntax:

- `()` is recognized as an empty list.
- `url()` is fully supported.
- `expression()`, `alpha(opacity=...)`, and `calc()` are supported (and left alone, except for interpolation).
- Interpolation is part of the parser, rather than being applied before parsing, so there should be far fewer bugs with it.
- CSS escapes within barewords are recognized (and ignored).
- `!important` may have whitespace after the `!`.

Glossing over a bad parse now spits out a deprecation warning, and will be removed entirely in 2.0.

Bug fixes

- Old-style pseudo-element selectors (`:before` and friends, written with only one colon) always stay at the end of the selector.
- The CSS3 `grayscale(90%)` function is now left alone, rather than being treated as a Sass function. (Previously, only unitless numbers would trigger this behavior.)
- Placeholder selectors (`%foo`) no longer end up in the output.
- Iterating over a list of lists with a single variable works (again).
- File path handling is much more polite with Windows directory separators.
- At-rules broken across several lines are now recognized correctly.
- `@for ... to` now excludes the upper bound.
- `@extend` no longer shuffles rules around, and should now produce rules in the same order as Ruby Sass. It also produces rules in the correct order when extending from the same rule more than once. Hopefully it's now correct, once and for all.
- Fixed a couple more Compass gradient bugs. Probably.

New features

- Compatibility with Python 3.2, allegedly.
- Support for building SVG font sheets from within stylesheets.
- Error messages have been improved once again: parse errors should be somewhat less cryptic, the source of mixins is included in a traceback, and missing closing braces are reported.

Backwards-incompatible changes

- Missing imports are now fatal.
- Much sloppy behavior or reliance on old xCSS features will now produce deprecation warnings. All such behaviors will be removed in pyScss 2.0.

Internals

- The C speedups module is now Unicode-aware, and works under CPython 3.
- There's no longer a runtime warning if the speedups module is not found.
- pyScss is now (a lot more) Unicode-clean; everything internally is treated as text, not bytes.
- Compiling the grammar is now much less painful, and doesn't require copy-pasting anything.
- Several large modules have been broken up further. `__init__` is, at last, virtually empty.
- All the built-in functions have been moved into built-in extensions.

4.5.7 1.2.0 (Oct 8, 2013)

This is a significant release that greatly increases compatibility with the reference compiler; in particular, the Sass port of Bootstrap now compiles.

There are a lot of changes here, so please feel free to report any bugs you see! The goal is 100% compatibility with the Ruby project.

Missing Sass features

- Dashes and underscores are treated as interchangeable in variable, function, and mixin names.
- Rule blocks in the form `background: red { ... }` are now supported.
- Colors are output as their shortest representation, and never as `hsl()`. The separate compiler options for compressing colors have been removed.
- The color modification functions (`adjust-color`, etc.) now work reliably.
- `transparent` is recognized as a color.
- Unrecognized units are now supported and treated as opaque.
- Arbitrary combinations of units (e.g., `px * px`) are supported for intermediate values. Unit cancellation now works reliably.
- Comparison and addition are now more in line with the Ruby behavior.
- `/` is now left untouched when it appears between literals, as in `font: 0 / 0`.
- `null` is supported.
- `zip()` is supported.
- `grayscale()` now knows it's also a CSS3 filter function, and won't be evaluated if its argument is a number.
- Slurpy arguments (`some-function($args...)`) are supported.
- `@extend` has been greatly improved: it eliminates common ancestors and works in many complex cases that used to produce strange results.
- Several Compass functions now adhere more closely to Compass's behavior. `linear-gradient()` is less likely to wreck valid CSS3 syntax.
- Compass's `e()`, `pow()`, `log()`, and `sqrt()` are now supported.

Bug fixes

- Interactive mode works. Again.
- Color names in strings and selectors are no longer replaced with hex equivalents.
- Unrecognized `@`-rule blocks such as `@keyframes` are left alone, rather than being treated like selectors.
- `@media` blocks aren't repeated for every rule inside.
- Pound-interpolation always drops quotes on strings.
- Single quoted strings no longer lose their quotes when rendered.
- `+ foo { ... }` is now recognized as a nested block, not an include.
- `color-stop()` and several proposed CSS4 functions no longer produce "unrecognized function" warnings.
- Several obscure bugs with variable scoping have been fixed, though a couple others remain.
- Several bugfixes to the C speedups module to bring it in line with the behavior of the pure-Python scanner.

New features

- Python 3 support. As a result, Python 2.5 no longer works; whether this is a bug or a feature is not yet clear.
- It's possible to write custom Sass functions in Python, though the API for this is not final.
- Experimental support for the map type and destructuring `@each`, both unreleased additions to the Ruby project.
- Support for the new string and list functions in Sass 3.3.
- Added `background-brushed`.

Backwards-incompatible changes

- Configuration via monkeypatching the `scss` module no longer works. Monkeypatch `scss.config` instead.
- `em` and `px` are no longer compatible.
- Unrecognized variable names are now a fatal error.

Internals

- No longer a single 5000-line file!
- Vastly expanded test suite, including some experimental tests borrowed from the Ruby and C implementations.
- Parser now produces an AST rather than evaluating expressions during the parse, which allows for heavier caching and fixes some existing cache bugs.
- The type system has been virtually rewritten; types now act much less like Python types, and compilation uses Sass types throughout rather than mixing Python types with Sass types.

4.5.8 1.1.5 (Feb 15, 2013)

- `debug_info` now properly produces rules that can be used by FireSass and Google Chrome SASS Source Maps.
- Improved memory usage for large sets of files to be used as sprites.

- Warns about IE 4095 maximum number of selectors.
- `debug_info` prints info as comments if specified as `comments`.
- Better handling of undefined variables.
- Added CSS filter functions and `skewX` `skewY`.
- Command line tool and entry point fixed.
- Fix cache buster URLs when paths already include queries or fragments.
- Hashable Values.

4.5.9 1.1.4 (Aug 8, 2012)

- Added `--debug-info` command line option (for *FireSass* output).
- Added compass helper function `reject()`.
- Added `undefined` keyword for undefined variables.

4.5.10 1.1.3 (Jan 9, 2012)

- Support for the new Sass 3.2.0 features (`@content` and placeholder selectors)
- Fixed bug with line numbers throwing an exception.

4.5.11 1.1.2 (Jan 3, 2012)

- Regression bug fixed from 1.1.1

4.5.12 1.1.1 (Jan 2, 2012)

- Added optional C speedup module for an amazing boost in scanning speed!
- Added `headings`, `stylesheet-url`, `font-url`, `font-files`, `inline-font-files` and `sprite-names`.

4.5.13 1.1.0 (Dec 22, 2011)

- Added `min()` and `max()` for lists.
- Removed exception raise.

4.5.14 1.0.9 (Dec 22, 2011)

- Optimizations in the scanner.
- Added `background-noise()` for compass-recipes support.
- `enumerate()` and `range()` can go backwards. Ex.: `range(3, 0)` goes from 3 to 0.
- Added line numbers and files for errors.
- Added support for *Firebug* with *FireSass*.

- `nth(n)` is round (returns the `nth mod len` item of the list).
- `--watch` added to the command line.
- Several bugs fixed.

4.5.15 1.0.8 (May 13, 2011)

- Changed source color (`$src-color`) default to black.
- Moved the module filename to `__init__.py` and module renamed back to scss.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search