

# MAKING DRUPAL SASSY

by Richard Lyon  
@richthegeek

# INSTALLATION PROCESS

- Download Libraries, Prepro and Sassy, and place them in your modules folder:
  - [drupal.org/project/libraries](http://drupal.org/project/libraries)
  - [drupal.org/project/prepro](http://drupal.org/project/prepro)
  - [drupal.org/project/sassy](http://drupal.org/project/sassy)
- Download PHPSass, and extract into your sites/all/libraries folder:
  - [github.com/richthegeek/phpsass](https://github.com/richthegeek/phpsass)
- Enable the Sassy module (this will auto-enable Libraries and Prepro)

`www ▶ drupal-7.12 ▶ sites ▶ all ▶ libraries ▶ phpsass`

The screenshot shows the Drupal module configuration page for the Sassy module. It is divided into two sections: 'OTHER' and 'THEMING TOOLS'. The 'OTHER' section contains two modules: 'Libraries' and 'Preprocessor', both of which are enabled. The 'THEMING TOOLS' section contains four modules: 'Sassy core SASS+SCSS compiler' (enabled), 'Sassy Bootstrap integration' (disabled), 'Sassy Compass integration' (disabled), and 'Sassy Foundation integration' (disabled). A 'Save configuration' button is located at the bottom of the page.

OTHER			
ENABLED	NAME	VERSION	DESCRIPTION
<input checked="" type="checkbox"/>	<b>Libraries</b>	7.x-1.0	Allows versioning of CSS files. Required by:
<input checked="" type="checkbox"/>	<b>Preprocessor</b>	7.x-0.4	Provides an interface for the SASS/SCSS preprocessor. Required by:

THEMING TOOLS	
ENABLED	NAME
<input checked="" type="checkbox"/>	<b>Sassy core SASS+SCSS compiler</b>
<input type="checkbox"/>	<b>Sassy Bootstrap integration</b>
<input type="checkbox"/>	<b>Sassy Compass integration</b>
<input type="checkbox"/>	<b>Sassy Foundation integration</b>

Save configuration

# PREPRO CONFIGURATION

- Prepro provides settings handling for all registered preprocessors, for simpler management.

Home » Administration » Configuration » Media

### FILETYPE-SPECIFIC SETTINGS

FORMAT	PRE-PROCESSOR	CACHING	CACHE LOCATION
.sass	Sassy PHPSass ▼	Cached, reprocessed on cache-clear ▼	public://prepro/
.scss	Sassy PHPSass ▼	Cached, reprocessed on cache-clear ▼	public://prepro/

▼ **SASSY PHPSASS**

Pass @warn and @debug to Watchdog  
Should the compiler pass results of @warn and @debug directives in the parsed files to the Drupal Watchdog system.

Include debug information in output  
Should the compiler include debugging text, usable by [Firebug](#) for enhanced debugging.

**Error reporting method**  
Watchdog: errors are recorded by the Watchdog to be viewed by an administrator. ▼  
How should the compiler record/display errors from processing?

**Output style**  
Nested: Each property+selector takes up 1 line, selector indentation reflects nesting depth. ▼  
What style of output should Sassy use.

Save Settings

# YOUR FIRST SCSS FILE

## + ERROR HANDLING

- Using SASS is now as easy as adding a .scss or .sass file to your CSS list:

```
7 stylesheets[all][] = css/colors.css
1 stylesheets[print][] = css/print.css
2
3 stylesheets[all][] = css/style.scss
4
5 regions[header] = Header
5 regions[help] = Help
```

- If you use “show on page” errors, errors appear before the page content:

**Error reporting method**

Show on page: an error message is shown on the page in compatible browsers.

How should the compiler record/display errors from processing?

Dashboard Content Structure Appearance People Me

```
Number must be a number: themes/bartik/css/style.scss::2
Source: color: 3 + false
```

# HISTORY

- Sassy is a system for allowing Drupal to compile SASS and SCSS files on the fly.
- Comprised of two modules and a library:
  - Prepro
  - Sassy
  - PHPSass (our fork of the PHamIP library)
- Started in August 2011 by Richard Lyon (@richthegeek) and Sebastian Siemssen (@thefubhy) due to poor existing Drupal modules. Originally it was just the Sassy module which included the parser and everything that is now in the Prepro module.
- Split into the Prepro + Sassy + PHamIP structure in January 2012. This allows greater modularity and made PHamIP less dependant on Drupal (it is now entirely independent).
- Renamed our PHamIP fork to PHPSass, to mark that it's no longer a branch and is purely focused on SASS parsing.

# WHAT IS PREPRO?

[HTTP://DRUPAL.ORG/PROJECT/PREPRO](http://drupal.org/project/prepro)

- Prepro is a middle-man tool designed to make the act of making a CSS pre-processor very easy to implement.
- It handles picking the right files and passing them to “registered” pre-processors.
- Output is optionally cached on a per-filetype and/or per-file basis .
- Settings for each filetype and pre-processor are handled without hassle.

# WHAT IS SASSY?

[HTTP://DRUPAL.ORG/PROJECT/SASSY](http://drupal.org/project/sassy)

- Sassy acts as a registered pre-processor for Prepro. It offers to process .sass and .scss files.
- It provides settings (via Prepro) that control the PHPSass library and error handling.
- Files are then parsed by the PHPSass library.
- Sassy provides hooks for custom functions and namespaces, which allows fully modular extensions such as Compass or Zurb Foundation.

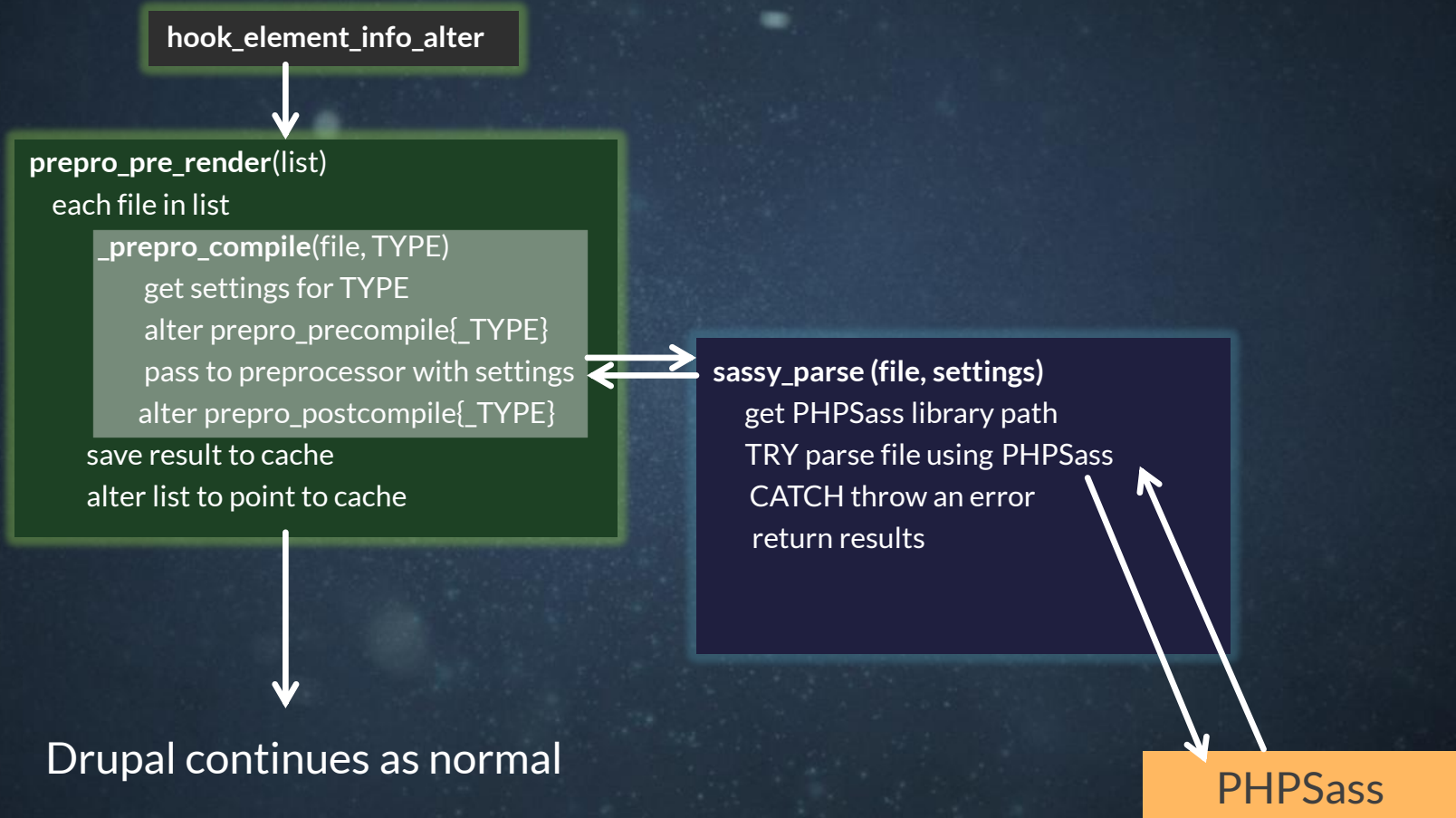
# WHAT IS PHPSASS?

[HTTPS://GITHUB.COM/RICHTHEGEEK/PHPSASS](https://github.com/richthegeek/phpsass)  
[@PHPSASS](#)

- PHPSass is a fork of the PHamlP library.
- It is a SASS compiler written entirely in PHP.
- Since forking, we've managed to fix almost all bugs that existed before, and implemented almost all missing features. The compiler is now:
  - Complete to v3.1.1.5 of the Ruby compiler
  - Tracking v3.2 development with a lag of about 2 weeks.
- We've stripped out large chunks of extraneous features for now – some of them will be added back in at a later date but we wanted to fix the language processing before anything else.



# HOW THEY WORK TOGETHER



# CREATE A PREPRO PREPROCESSOR

Drupal

```
function sassruby_css_preprocessor_info() {
  return array(
    'sassruby' => array(
      'label' => 'SASS Ruby parser',
      'callback' => 'parse',
      'filetypes' => array('sass', 'scss')
    )
  );
}

function sassruby_parse($file, $settings) {
  $local += array('style' => 'nested');
  exec($file['extension'] . ' --style=' . $local['style'] . ' ' . $file['data'], $output);
  return implode($output);
}

function sassruby_css_preprocessor_settings_form($form, $form_state) {
  $local += array('style' => 'nested');
  $form['style'] = array(
    '#type' => 'select',
    '#title' => 'Output style',
    '#description' => t('What style of output should the parser use.'),
    '#options' => array(
      'nested' => 'Nested: Each property+selector takes up 1 line, selector indentation ...',
      'expanded' => 'Expanded: Each property+selector takes up 1 line, no selector indentation.',
      'compact' => 'Compact: Each selector takes up 1 line with properties on the same line.',
      'compressed' => 'Compressed: Almost no whitespace, designed to be as space-efficient ...',
    ),
    '#default_value' => $local['style'],
  );
  return $form;
}
```

# HOW TO ADD A LIBRARY

## A CASE STUDY OF COMPASS INTEGRATION

- Compass contains two groups of data:
  - ~80 .scss files comprising most functionality
  - ~20 .rb files containing functions that are not implementable with SASS
- The SASS files aren't a problem, and are a great test of the compiler. However, the directory structure is somewhat verbose so we need to tell PHPSass how to find the files with a namespace. We add the “compass” namespace via *hook\_sassy\_resolve\_path\_compass*, so we can do:
  - @import “compass/compass”;
  - @import “compas/css3”;
- The ruby files are a sticking point – we can't run Ruby in PHP so we need to convert the functions and make PHPSass aware of them. We add the functions via *hook\_sassy\_functions*.

# HOW TO ADD A LIBRARY

## PATH LOOKUPS AND NAMESPACES

- Whenever PHPSass tries to find a file and it cannot, it iterates through a list of load path functions.
- Functions receive a filepath and should return an array of proper filepaths.
- Sassy extends this so your module can respond to a single namespace, for proper segregation of responsibilities.

PHPSass

```
$parser = new SassParser(array(
    'load_path_functions' => array(
        'my_custom_function'
    ));

function my_custom_function($file) {
    return array("/var/www/$file");
}
```

Drupal

```
function hook_sassy_resolve_path_foo($file) {
    return array("~/stylesheets/$file");
}
```

Result

```
@import    foo/styles;
PHPSass:   /var/www/foo/styles
Sassy:     ~/stylesheets/styles
```

# HOW TO ADD A LIBRARY

## CUSTOM FUNCTIONS

- Whenever PHPSass tries to call a function, it looks up a list of functions passed into the options.
- Functions receive arguments in string form, and we use PHP5 Reflections to verify argument counts and names where possible.
- Sassy provides a near direct hook.

PHPSass

```
$parser = new SassParser(array(
    'functions' => array(
        'reverse' => 'strrev'
    ));
```

Drupal

```
function hook_sassy_functions() {
    return array(array(
        'name' => 'reverse',
        'callback' => 'strrev'
    ));
}
```

Result

```
content:      reverse('foobar');
content:      baroof;
```

# HOW TO ADD A LIBRARY

## CUSTOM FUNCTIONS: ARGUMENT NAMING + DEFAULTS

PHPSass

```
function x($a = 'foo', $b = 'bar') {  
    return $one . ' ' . $two;  
}
```

- PHPSass uses the PHP5 Reflections API to figure out if we can use named arguments with defaults.
- This means your PHP functions act in a manner indistinguishable from your SASS functions.
- Compatible with PHP5.0, like the rest of PHPSass.
- For this reason, you should consider naming your arguments in obvious ways (bad example).

SCSS

```
default: x();  
onetwo: x('one', 'two');  
onebar: x('one');  
footwo: x($b: 'two');  
reverse: x($b: 'rev', 'erse');
```

CSS

```
default: foo bar;  
onetwo: one two;  
onebar: one bar;  
footwo: foo two;  
reverse: erse rev;
```