

Using Sass

Using Sass

About Sass

Sass is the acronym for "Syntactically Awesome Style Sheets". It is a language that you can use to build your CSS files.

While remaining CSS3-compatible, it brings a lot of features that make it easier to create consistent CSS rules with less work, most notably less copy-pasting: nesting, variables, classes, control directives (if, for, each, while), etc.

To achieve this, Sass requires the use of a preprocessor: it turns your Sass files into CSS files that all browsers can read.

Sass has two syntaxes it can use:

- SCSS syntax, or "Sassy CSS":
 - **The newer syntax, and the one PrestaShop uses.**
 - Files use the `.scss` extension.
 - Can use semicolons and parenthesis.
 - Extension of CSS3.
- SASS syntax, or "indented syntax":
 - Older syntax, less used.
 - Files use the `.sass` extension.
 - Relies on tab indentation, just as in Python.
 - No semicolon, no parenthesis.
 - Properties must start with a newline.

You can see the difference between both syntaxes here:

SCSS syntax

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

SASS syntax

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

Installation

Sass (and Compass in the first place, see below) require the installation of the Ruby language on your machine.

Here are some installers:

- Windows: Download <http://rubyinstaller.org/> and launch it.
- Mac OS X: Install Homebrew (<http://brew.sh/>), then type "brew install ruby".
- Linux systems: In the command line, type "sudo apt-get install ruby1.9.1".

Once Ruby is installed on your machine, install Compass – which will in turn install Sass. Go to your Ruby command line and type "gem install compass".

Starting a project project

You are now ready to create your first project:

1. Go to the `/themes` folder of your local installation of PrestaShop.
2. Open a Ruby command line window for this folder, and type `"compass create"`.
Compass will automatically detect or create the `config.rb` configuration file that is necessary for the compilation of your project.

To make sure that `.scss` files are automatically compiled, you can type this Ruby command: `"compass watch"`.

Not a command line fan?

If you would rather not spend your time in the command line, Scout is the cross-platform application for you: it is a self-contained Ruby environment that gives you easy access to Compass and Sass.

Download it here: <http://mhs.github.io/scout-app/>

After you have installed it, you must:

1. Create your project at the root of your theme's folder.
2. Indicate the folder of your `.scss` files (Input folder).
3. Indicate the folder for your CSS files (Output folder).

Sass syntax

Comments

There are two ways to add a comment to a Sass file.

The first one enables you to add comments without having them used in the final CSS file:

```
// This comment is not used in the CSS file.  
a { color: green; }
```

The second one uses the regular CSS syntax, and enables you to add comments that WILL be used in the final CSS file:

```
/* This comment is used in the CSS file. */  
a { color: green; }
```

Nesting

Sass enables you to nest blocks in order to define rules that apply only within that selector:

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  li { display: inline-block; }  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

Result:

```
nav ul {margin: 0; padding: 0; list-style: none;}
nav li {display: inline-block;}
nav a {display: block; padding: 6px 12px; text-decoration: none;}
```

Parent selector

Sass enables you to reference the parent selector in a nested block, using &:

```
a {
  font-weight: bold ;
  &:hover{
    color: red ;
  }
  li & {
    font-weight: normal;
  }
}
```

Result:

```
a {font-weight: bold ;}
a:hover {color: red ;}
li a {font-weight: normal;}
```

Namespace properties

CSS can use namespace-like properties, such as font-family, font-size, etc.

You can use nesting to indicate properties within a given "namespace":

```
p {
  font: {
    family: arial;
    size: 1.1em;
    weight: bold;
  }
}
```

Result:

```
p {
  font-family: arial;
  font-size: 1.1em;
  font-weight: bold;
}
```

CSS output formats

Sass allows 4 different kind of generated CSS.

Nested format:

```
#main {
  color: #fff;
  background-color: #000; }
#main p {
  width: 10em; }
```

Expanded format:

```
#main {
  color: #fff;
  background-color: #000;
}
#main p {
  width: 10em;
}
```

Compact format:

```
#main { color: #fff; background-color: #000; }
#main p { width: 10em; }
```

Compressed format:

```
#main{color:#fff;background-color:#000}#main p{width:10em}
```

SassScript

Sass can make use of a scripting language called SassScript. It makes it possible to use variables, calculation and additional functions. It can be used for any property value.

Variables

Variables must be defined with the `$` prefix:

```
$blue: #3bbfce;
$margin: 16px;
.content-navigation {
  border-color: $blue;
  color: darken($blue, 9%);
}
.border {
  padding: $margin/2;
  margin: $margin/2;
  border-color: $blue;
}
```

You can change the value of a variable, or have it changed only if the variable does not exist yet or is empty by using `!default`.

For instance:

```
$content: "Init Value";
$content: "Init if no value" !default;
$new_content: null;
$new_content: "Init new if no value" !default;
#main {
  content: $content;
  new-content: $new_content;
}
```

Result:

```
#main {content: "Init Value"; new-content: "Init new if no value";}
```

SassScript supports 6 types of variables:

- numbers (e.g. 1.2, 13, 10px)
- strings of text, with and without quotes (e.g. "foo", 'bar', baz)
- colors (e.g. blue, #04a3f9, rgba(255, 0, 0, 0.5))
- booleans (e.g. true, false)
- nulls (e.g. null)
- lists of values, separated by spaces or commas (e.g. 1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
- maps from one value to another (e.g. (key1: value1, key2: value2))

When using a string value, you can use `#{ }` to unquote quoted strings, making it easier to use in some cases, for instance as a mixin selector:

```
@mixin warn-message($selector) {
  body.warn #{ $selector }:before {
    content: "This is a warning !";
  }
}
@include warn-message(".header");
```

Result:

```
body.warn .header:before {content: "This is a warning !";}
```

You can also use the `quote()` and `unquote()` functions:

```
$family: unquote("Droid+Sans");
@import url("http://fonts.googleapis.com/css?family=#{ $family }");
```

Result:

```
@import url("http://fonts.googleapis.com/css?family=Droid+Sans");
```

Operators

Your SCSS files can directly use calculations and comparators:

- +, -, *, /, % and the <, >, <=, => number comparators.
- + to concatenate two strings.
- and, or, andnot for boolean variables.
- == and != for all variable types.

Sass can use the / operator to separate two numerical values, but Sass can use it to divide numbers. For instance:

```
p {
  font: 10px/8px;           // Plain CSS, no division
  $width: 1000px;
  width: $width/2;         // Uses a variable, does division
  width: round(1.5)/2;     // Uses a function, does division
  height: (500px/2);       // Uses parentheses, does division
  margin-left: 5px + 8px/2px; // Uses +, does division
}
```

...is compiled into:

```
p {
  font: 10px/8px;
  width: 500px;
  height: 250px;
  margin-left: 9px; }
```

As usual, use parenthesis to handle priorities:

```
p {
  cursor: e + -resize;
  font-family: sans- + "serif";
  margin: 3px + 4px auto;
  width: (1em + 2em) * 3;
}
```

Result:

```
p {cursor: e-resize; font-family: sans-serif; margin: 7px auto; width: 9em;}
```

You can also use several numerical functions:

- `percentage($value)` - Converts a unitless number to a percentage.
- `round($value)` - Rounds a number to the nearest whole number.
- `ceil($value)` - Rounds a number up to the next whole number.
- `floor($value)` - Rounds a number down to the previous whole number.
- `abs($value)` - Returns the absolute value of a number.
- `min($numbers...)` - Finds the minimum of several numbers.
- `max($numbers...)` - Finds the maximum of several numbers.

Lists

Sass uses lists for values such as `margin: 10px 15px 0 0` or `font-face: Helvetica, Arial, sans-serif`.

Lists are a series of values, separated with space or commas.

A list can contain one or more lists. For instance, "1px, 2px, 5px 6px" is made of three elements:

- 1px
- 2px
- 5px 6px

The third element is itself a list of elements.

You have access to several list functions:

- `length($list)` - Returns the length of a list.
- `nth($list, $n)` - Returns a specific item in a list.
- `join($list1, $list2, [$separator])` - Joins together two lists into one.
- `append($list1, $val, [$separator])` - Appends a single value onto the end of a list.
- `zip($lists...)` - Combines several lists into a single multidimensional list.
- `index($list, $value)` - Returns the position of a value within a list.

The @ directives

Sass supports all of CSS3's @ rules, and adds a few more features such as:

- Control directives for the CSS compilation.
- Use of mixins.

@import

You can create files that contain the Sass instructions that you can include in other files. These "imported" files must have an underscore ("_") at the beginning of their names so as to not be directly compiled into CSS.

For instance:

```
_partial.scss

html, body, ul, ol {
  margin: 0;
  padding: 0;
}
```

Importing `_partial.scss` into `global.scss`:

```
@import 'partial';
body {
  font-size: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

@import automatically searches files with a `.scss` or `.sass` extension.

You can import several files with a single call: `@import "rounded-corners", "text-shadow"`.

You can import a file from a nested block:

```
/* global.scss */
#main {
  // imports _example.scss
  @import "example";
}
```

@mixin

Mixins make it possible to group CSS properties that you can reuse as many times as necessary. They can even include other mixins.

SCSS example:

```
/* File _mixin.scss */
@mixin class-inclusion {
  th {
    text-align: center;
    font-weight: bold;
  }
  th, td {
    padding: 2px;
  }
}
@mixin highlighted-background {background-color: #fc0;}
@mixin header-text {font-size: 20px;}
// mixin with other mixins
@mixin compound {
  @include highlighted-background;
  @include header-text;
}
```

Mixins are then used with the @include directive:

```
/* File global.scss */
@import "mixin.scss";
.sassEx {@include class-inclusion;}
.mycompound {@include compound;}
```

Mixins can have parameters:

```
/* File _mixin.scss */
// mixins with parameters and default value
@mixin sexy-border($color, $width: 1in) {
  border: {
    color: $color;
    width: $width;
    style: dashed;
  }
}
// mixin with unknown number of arguments
@mixin box-shadow($shadows...) {
  -moz-box-shadow: $shadows;
  -webkit-box-shadow: $shadows;
  box-shadow: $shadows;
}
// mixin with unknown number of arguments and other mixin
@mixin bold-box-shadow($shadows...) {
  font-weight: bold;
  @include box-shadow($shadows...);
}
```

Used in:


```

/* File global.scss */
@import "mixin.scss";
// arguments with default values can be omitted
.box { include sexy-border(blue); }
// with explicit keyword name, arguments can be passed in any order
.bigbox { include sexy-border($width: 3in, $color:blue); }
// use mixin with unknown number of arguments
.shadows { @include box-shadow(0px 4px 5px #666, 2px 6px 10px #999); }
// use mixin with variable and unknown number of arguments
$param: 0px 4px 5px #666, 2px 6px 10px #999;
.shadows { @include box-shadow($param...);

```

You can directly pass a style block to a mixin in order to place in the defined style, using @content.

SCSS example:

```

@mixin apply-to-ie6-only {
  * html {
    @content;
  }
}
@include apply-to-ie6-only {
  #logo {
    background-image: url(/logo.gif);
  }
}

```

CSS result:

```

* html #logo {
  background-image: url(/logo.gif);
}

```

Note that variables are evaluated within the block where they are defined:

SCSS example:

```

$color: white;
@mixin colors($color: blue) {
  background-color: $color;
  @content;
  border-color: $color;
}
.colors {
  @include colors {color: $color;}
}

```

CSS result:

```

.colors {
  background-color: blue;
  color: white;
  border-color: blue;
}

```

@media

Sass allows the nesting of definition blocks from a @media directive.

@media directives can be nested, and they can use SassScript.

For instance, in SCSS:

```
.sidebar {
  width: 300px;
  @media screen and (orientation: landscape) {
    width: 500px;
  }
}
@media screen {
  .sidebar {
    @media (orientation: landscape) {
      width: 500px;
    }
  }
}
$media: screen;
$feature: -webkit-min-device-pixel-ratio;
$value: 1.5;
@media #{$media} and ($feature: $value) {
  .my_sidebar {
    width: 500px;
  }
}
```

CSS compilation result:

```
.sidebar {width: 300px; }
@media screen and (orientation: landscape) {
  .sidebar {width: 500px; }
}
@media screen and (orientation: landscape) {
  .sidebar {width: 500px; }
}
@media screen and (-webkit-min-device-pixel-ratio: 1.5) {
  .my_sidebar {width: 500px; }
}
```

@extend

Inheritance makes it possible to import styles from one CSS selector to another.

SCSS examples:

```
.message {
  border: 1px solid #ddd;
  padding: 20px;
  color: #222;
}
.success {
  @extend .message;
  border-color: green;
}
.error {
  @extend .message;
  border-color: red;
}
```

CSS compilation result:

```
.message, .success, .error {
  border: 1px solid #ddd;
  padding: 20px;
  color: #222;
}
.success {
  border-color: green;
}
.error {
  border-color: red;
}
```

You can also chain inheritance:

SCSS example:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError {
  @extend .error;
  border-width: 3px;
}
.criticalError {
  @extend .seriousError;
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%;
}
```

CSS compilation result:

```
.error, .seriousError, .criticalError {
  border: 1px #f00;
  background-color: #fdd;
}
.seriousError, .criticalError {
  border-width: 3px;
}
.criticalError {
  position: fixed;
  top: 10%;
  bottom: 10%;
  left: 10%;
  right: 10%;
}
```

To create a class that would only be available through inheritance, you can use % instead of @. %-using classes are not present in the generated CSS file.

SCSS example:

```
#main a%bigblue {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
.notice {
  @extend %bigblue;
}
```

CSS compilation result:

```
#main a.notice {
  color: blue;
  font-weight: bold;
  font-size: 2em;
}
```

By default, you cannot use `@extend` with a non-existing selector: it gives a compilation error. To use a selector that may not yet exist in certain conditions, you should use the `!optional` flag:

```
a.important {
  @extend .notice !optional;
}
```

Likewise, you cannot use `@extend` in all situations, for instance when using CSS directives such as `@media`.

You must declare your selectors and use them within a single directive.

Example in SCSS:

```
@media print {
  .error {
    border: 1px #f00;
    background-color: #fdd;
  }
  .seriousError {
    @extend .error;
    border-width: 3px;
  }
}
```

Non-valid example in SCSS:

```
.error {
  border: 1px #f00;
  background-color: #fdd;
}
@media print {
  .seriousError {
    // INVALID EXTEND: .error is used outside of the "@media print" directive
    @extend .error;
    border-width: 3px;
  }
}
```

@warn and @debug

Two directives can help you debug your Sass files:

- @debug displays the result of a SassScript expression.
- @warn displays information messages during the CSS file compilation.

SCSS Example:

```
@debug 10em + 12em;
```

Displays:

```
Line 1 DEBUG: 22em
```

Other SCSS example:

```
@mixin adjust-location($x, $y) {  
  @if unitless($x) {  
    @warn "Assuming #{$x} to be in pixels";  
    $x: lpx * $x;  
  }  
  @if unitless($y) {  
    @warn "Assuming #{$y} to be in pixels";  
    $y: lpx * $y;  
  }  
  position: relative; left: $x; top: $y;  
}
```

Control instructions: @if, @for, @each, @while

These instructions are mainly used in mixins of Sass libraries such as Compass.

SCSS example:

```
$type: monster;  
p {  
  @if $type == monster {  
    color: green;  
  } @else {  
    color: black;  
  }  
}  
@for $i from 1 through 3 {  
  .item-#{$i} {width: 2em * $i;}  
}  
@each $animal in puma, sea-slug, egret, salamander {  
  .#{$animal}-icon {  
    background-image: url('/images/#{$animal}.png');  
  }  
}  
$i: 6;  
@while $i > 0 {  
  .item-#{$i} {width: 2em * $i;}  
  $i: $i - 2;  
}
```

CSS compilation result:

```
p {color: green;}
.item-1 {width: 2em;}
.item-2 {width: 4em;}
.item-3 {width: 6em;}
.puma-icon {background-image: url('/images/puma.png');}
.sea-slug-icon {background-image: url('/images/sea-slug.png');}
.egret-icon {background-image: url('/images/egret.png');}
.salamander-icon {background-image: url('/images/salamander.png');}
.item-6 {width: 12em;}
.item-4 {width: 8em;}
.item-2 {width: 4em;}
```

@function

You can create your own functions and use it within your files. Just make sure that its name does not conflict with the evolution of Sass and Compass.

SCSS example:

```
$grid-width: 40px;
$gutter-width: 10px;
@function grid-width($n) {
  @return $n * $grid-width + ($n - 1) * $gutter-width;
}
#sidebar {width: grid-width(5);}
```

CSS result:

```
#sidebar {width: 240px;}
```