

Guide for Dynamic CSS

(version 2.0)

Kepler Gelotte
[Neighbor Webmaster](#)
Kepler@neighborwebmaster.com

Contents

Guide for Dynamic CSS	1
Contents	2
Tips and Usage.....	3
Customizing	4
@include	5
Variables	6
Using PHP Variables	7
Expressions	9
Conditional Logic	10
Modifying Headers	11

Tips and Usage

- ☞ These scripts may not work on older versions of Apache (i.e. apache 1.x).
- ☞ You don't need to modify any of your existing CSS, JavaScript, or image files to use these scripts.
- ☞ Place these three files in any directory containing CSS:
 - css/.htaccess
 - css/css-filter-start.php
 - css/css-filter-end.php
- ☞ Place these two files in any directory containing JavaScript:
 - javascript/.htaccess
 - javascript/gzip-js.php
- ☞ Note that you may have to turn off `comment_compress` in `css/css-filter-start.php` if you are using browser specific hacks (e.g. Holly Hack). See the "Customizing" section below. These settings can now be overridden in the CSS URL.
- ☞ When using the expanded syntax for CSS below, statements must be separated by either a new line and/or a semicolon (;).
- ☞ These scripts set the expires header to 1 year by default. To force a download of a modified CSS, JavaScript, or image file you need to either change the name of the file or add a parameter as in:


```
<link type="text/css" rel="stylesheet" src="css/main.css?version=1.1" />
```

-or-

```
<link type="text/css" rel="stylesheet" src="css/main_1_1.css" />
```
- ☞ When debugging, add `"?compress_comments=false"` to the end of your CSS URL. All error messages are put into the CSS as comments. Leaving the default settings, these comments will be stripped out. Here is an example:


```
<link type="text/css" rel="stylesheet" src="css/main.css?compress_comments=false" />
```

Customizing

At the top of `css-filter-start.php` script are variables that control the behavior of the Dynamic CSS replacement. Here is a description of the current variables:

\$debug – setting to true will output debug comments of the form `/* debug */` in the output CSS. This setting is useful if trying to debug an issue with this filter. It defaults to false.

\$allow_eol_comments – if set to true, will allow comments starting with `//` and run to the end of line. It defaults to true.

\$compress – if set to true will remove extra whitespace. It will leave a newline after each definition (after each `}` curly brace) for readability. This setting defaults to true.

\$compress_comments – removes all comments (regular and end-of-line if enabled). This defaults to true.

\$handle_pngs – will add the alpha filter if the browser's UserAgent identifies itself as IE5.5/IE6. For older versions of IE the `.png` will be replaced with `.gif` or `.png8` (see `$convert_to_png8`). If the `.gif` or `.png8` file doesn't exist, it will be created on the server as long as you have the GD2 image library installed on your server. This setting defaults to true.

\$use_alpha_filter – if you want to use Internet Explorer's alpha filter for versions IE5.5/IE6. If you would rather replace the `.png` with a `.gif/.png8` the set this value to false. This setting defaults to true.

\$convert_to_png8 – `png8` is an indexed 8 bit color version of the PNG format. It is similar to GIF and displays with transparency in older browsers. It does not support alpha transparency though. This setting defaults to true.

- You can now override these settings by passing the parameters on the CSS URL set as true or false. For example:

```
<link type="text/css" rel="stylesheet" src="css/main.css?compress_comments=false" />
```

@include

```
@include 'file.css' [;]
@include url('file.css') [;]
```

Both forms are equivalent. The syntax mimics the @import command. This command is like @import except it includes the CSS on the server. The @include may also be on any line within your CSS file. The text from the included CSS file will be added in place. You can nest @includes as well.

main.css:

```
body {
    color: #000;
    background-color: #CCC;
    @include 'second.css'
}
```

second.css:

```
@include 'third.css'
    padding: $padding;
```

third.css:

```
    text-decoration: none;
set $padding = 5em;
```

Will send the following to the browser when main.css is requested:

```
body {
    color: #000;
    background-color: #CCC;
    padding: 5em;
    text-decoration: none;
}
```

Variables

```
set $name = value [;]  
eval $name = expression [;]
```

Variables set using the set command are not PHP variables. They are not typed and behave just as string. Do **not** surround the value with quotes unless you want the quotes to be included in the variable substitution.

Use the set command to set a static string to a variable. Use the eval command to assign the output of the expression to a variable. The eval command is a good way to bridge to PHP since PHP's variables like \$_GET, \$_POST, \$_REQUEST, \$_SERVER, \$_COOKIES are not directly usable as variables in the CSS file.

The optional semicolon at the end of the set and eval statements will be removed.

Variables are substituted before expressions are evaluated (via PHP). Variables can also be added anywhere in the CSS and will be expanded when output.

An example setting a variable to a static value:

```
set $color = #0fd98e;
```

An example setting a variable based on the output from a PHP eval():

```
eval $user_agent = $_SERVER['HTTP_USER_AGENT'];
```

IMPORTANT: The above variable \$user_agent does not have surrounding quotes, so when used in an expression, you must add the quotes:

```
if ( strstr( '$user_agent', 'Mac' ) !== false )
```

Using PHP Variables

```
$_GET  
$_POST  
$_REQUEST  
$_COOKIES
```

Passing \$_GET variables to CSS:

```
<link type="text/css" rel="stylesheet" src="css/main.css?theme=blue" />
```

main.css:

```
#theme {  
    color: black;  
    if ( isset($_GET['style']) && $_GET['style'] == 'blue' )  
        background-color: blue;  
    else  
        background-color: white;  
    endif  
}
```

Variables and variable substitution are not handled by PHP so the following will **not** work:

```
body  
{  
    background-color: $_GET['background_color'];  
    color: $_GET['color'];  
    font: normal 75%/1.3em Verdana, Geneva, Helvetica, sans-serif;  
    text-align: center;  
}
```

By setting a variable using the eval command, the following **will** work:

```
eval $background_color = $_GET['background_color'];  
eval $color = $_GET['color'];  
body  
{  
    background-color: $background_color;  
    color: $color;  
    font: normal 75%/1.3em Verdana, Geneva, Helvetica, sans-serif;  
    text-align: center;  
}
```

Since expressions are evaluated using PHP, you **can** use the server variables in if and elif /elseif expressions:

```
body
{
    color: #000;
    if ($_GET['theme'] == 'blue')
        background-color: #03C;
    else
        background-color: #CCC;
    endif
}
```


Expressions

```
eval expression [;]
eval $variable = expression [;]
if ( expression ) [;]
elseif ( expression ) [;]
elif ( expression ) [;]
```

Expressions are evaluated using PHP. This means you have the full PHP syntax available to you in these statements.

The first form of the eval command passes off the expression to PHP for execution but ignores any result. The second form not only passes an expression off to the PHP engine for evaluation but also assigns any returned value to \$variable cast as a string.

Expressions in conditional statements are passed to PHP and the returned result is cast to the boolean type.

Conditional Logic

```
if ( expression ) [;]
elif ( expression ) [;] –or– elseif ( expression ) [;]
else [;]
endif [;]
```

Conditional logic works similar to most languages. elif and elseif are the same function, use whichever is more familiar to you.

An important note about how the expressions are evaluated – first any variables are substituted using a text replacement. Next the entire expression is handed off to PHP's eval command and the result is typed to a boolean. This means variables need to be quoted in the expression if they are to be treated as strings. Here are some examples to make it clear:

```
set $var_no_quotes = border: solid #000 1px;
```

```
if ( $var_no_quotes == 'border: solid #000 1px' )
```

FAILS because PHP is given

```
eval( return (boolean)( border: solid #000 1px == 'border: solid #000 1px' )
```

```
if ( '$var_no_quotes' == 'border: solid #000 1px' )
```

Works as expected

```
eval $quoted_user_agent = "" . $_SERVER['HTTP_USER_AGENT'] . "";
```

```
if ( strstr( $quoted_user_agent, 'Mac' ) === false )
```

Works as expected

Modifying Headers

```
header header_information [;]
expires
    immediate | yesterday
    now
    n minute[s]
    n hour[s]
    n day[s]
    n week[s]
    n month[s] (approximate)
    n year[s]
```

You can modify any header sent to the browser using the first form. An example using variable substitution and conditional logic:

```
eval $agent = $_SERVER['HTTP_USER_AGENT'];
set $header = Vary: Accept-Encoding;
if ( stristr( '$agent', 'msie' ) != false )
    header $header;
endif
```

The expires command sets the expiration of the CSS file. This tells the browser when to next download this file. Note if you set a far expires date (e.g. expires 1 year), you cannot force a refresh by setting an expires immediate. This is because the browser may not even request that CSS file until it expires (e.g. a year later).