Cascading Style Sheets

Overview

CSS stands for Cascading Style Sheet. HTML tags specify the graphical flow of the elements, be it text, images or flash animations, on a webpage. CSS allows us to define the appearances of those HTML tags with their content, somewhere, so that other pages, if want be, may adhere to. This brings along consistency throughout a website. The cascading effect stipulates that the style of a tag (parent) may be inherited by other tags (children) inside it.

Professional websites separate styling from content. There are many reasons for this, the most obvious (to a developer) being the ability to control the appearance of many pages by changing one file. Styling information includes: fonts, backgrounds, images (that recur on every page), position and dimensions of elements on the page. Your HTML file will now be left with: header information; a series of elements; the text of your website. Because you are creating a Joomla! template, you will actually have: some header information, PHP code to request the rest of the header information, a series of elements, PHP code to request each module position, and PHP code to request the main content.

Style information is coded in CSS and usually stored in files with the suffix .css. A webpage contains a link to the associated .css file so a browser can find the appropriate style information to apply to the page. CSS can also be placed inside a HTML file between <style type="text/css"></style> tags. This is, however, discouraged as it is mixing style and content elements which can make future changes more difficult.

Implementation

Definitions for this section:

External CSS files
    using <link> in the <head>
    Document head CSS
        using <style> in the <head>
    Inline CSS
        using the style attribute on a tag, i.e.

Guidelines

1. External CSS files should be used in preference to document head CSS and document head CSS should be used in preference to inline CSS.
2. CSS files MUST have the file extension .css and should be stored in the relevant includes directory in the site structure, usually /style/.
3. The file size of CSS files should be kept as low as possible, especially on high demand pages.
4. External CSS must be linked to using the <link> element which must be placed

in the head section of the document. This is the preferred method of using CSS. It offers the best experience for the user as it helps prevent FOUC (flash of unstyled content), promotes code reuse across a site and is cacheable.
5. External style sheets should not be imported (i.e. using @import) as it impairs caching. In IE @import behaves the same as using <link> at the bottom of the page (preventing progressive rendering), so it's best not to use it. Mixing <link> and @import has a negative effect on browsers' ability to asynchronously download the files.
6. Document head CSS may be used where a style rule is only required for a specific page.
7. Inline styles should not be used.
8. Query string data (e.g. "style.css?v=0.1") should not be used on an external CSS file. Use of query strings on CSS files prevents them from caching in some browsers. Whilst this may be desirable for testing, and of course may be used for that, it is very undesirable for production sites.

Directory & Files

Convention places CSS files within a directory named css inside the template directory. While developers are not restricted to this convention, we do recommend it as it helps keep the layout and structure of HUBzero templates consistent. A developer from one project will instantly know where to find certain files and be familiar with the directory structure when working on a project originally developed by someone else.

There are a handful of common CSS files found among most HUBzero. While none of these are required, it is encouraged to follow the convention of including them as it promotes consistency among HUBzero templates and comes with the advantage that certain files, such as main.css are auto-loaded, thus reducing some work on the developer's part.

Here's the standard directory and files for CSS found in a HUBzero template:

```
/hubzero
  /templates
    /{TemplateName}
      /css
        error.css
        browser/ie7.css
        browser/ie8.css
        browser/ie9.css
        main.css
        print.css
        component.css
```

File details:

error.css
: This is the primary stylesheet loaded by error.php.

ie8.css
: Style fixes for Internet Explorer 8.

ie7.css
: Style fixes for Internet Explorer 7.

ie9.css
: Style fixes for Internet Explorer 9.

main.css
: This is the primary stylesheet loaded by index.php. The majority of your styles will be in here.

print.css
: Styles used when printing a page.

component.css
: This file is meant to be included **before** any other CSS file. Its purpose is to reduce browser inconsistencies in things like default line heights, margins and font sizes of headings, and so on.

Bootstrap

Several bootstrap styles are available in the core, broken into individual stylesheets to make it easier for you to decide what styles you do and do not want to incorporate into your template.

The bootstrap stylesheets can be found in the /media/system/css directory and can be linked to or imported like any other stylesheet. However, for sake of site performance, we recommend using the HubzeroDocumentAssets::getSystemStylesheet() method. This method accepts wither a comma-separated string or array of core stylesheets to include and then compiles them into a single file with comments and white-space stripped out. The resulting file is saved in the cache with a timestamp. Should any of the core files change, the resulting compiled stylesheet will automatically be updated. This has two immediate advantages of 1) fewer http requests (improves page load time) and 2) ensures browsers re-cache the CSS whenever it has changed.

Example usage:

```
<link rel="stylesheet" type="text/css" media="screen" href="<?php
 echo HubzeroDocumentAssets::getSystemStylesheet(array(
 'reset',
 'fontcons',
 'columns',
 'notifications',
 'pagination',
```

```
 'tabs',
 'tags',
 'comments',
 'voting',
 'layout'
)); ?>" />
```

reset.css

> This file is meant to be included **before** any other CSS file. Its purpose is to reduce browser inconsistencies in things like default line heights, margins and font sizes of headings, and so on.
>
> The reset styles given here are intentionally very generic. There isn't any default color or background set for the <body> element, for example. Colors and any other styling should be addressed in the template's primary stylesheet after loading reset.css.

fontcons.css

> This is a custom created icon (dingbat) font used for many of the icons found throughout a hub.

columns.css

> This sets up basic structure for generating layouts that use columns. It supports up to twelve columns and any combination there in. See usage.

notifications.css

> Default styles for warning, error, help, and info messages.

pagination.css

> Basic styling for pagination.

tabs.css

> Default styles for a menu (list) displayed as tabs.

tags.css

Tag styles. Tags are used frequently throughout a hub and this stylesheet helps ensure the look consistent.

comments.css

Comments appear on many items such as KB articles, Questions and Answers, Support tickets, Forums, Blog posts, and more. This is a stylesheet for handling basic layout and styles of a list of (nested) comments and the form for submitting comments.

voting.css

Basic styles for thumbs-up and thumbs-down voting buttons.

layout.css

Default styles for containers, result lists, and other basic structural items used frequently in a hub.

Typical main.css Structure

main.css controls base styling for your HUB, which is usually further extended by individual component CSS.

We took every effort to organize the main.css in a manner allowing you to easily find a section and a class name to modify. E.g. if you want to change the way headers are displayed, look for "headers" section as indicated by CSS comments. Although you can modify all existing classes, depending on your objectives, it is recommended to avoid modifications to certain sections, as indicated below. While you can add new classes as needed, we caution strongly about removing or renaming any of the existing IDs and classes. Many HUBzero components take advantage of these code styles and any alterations made risk breaking the template display.

Some sections that you are likely to modify:

```
Body - may want to change site background or font family.
Links - pick colors for hyperlinks
Headers - pick colors and font size of headings
```

```
Lists - may want to change general list style
Header - you will definitely want to change this
Toolbar - display of username, login/logout links etc.
Navigation - display of main menu
Breadcrumbs - navigation under menu on secondary pages
Extra nav - links that appear on the right-
hand side in multiple components
Footer
```

Sections where you would want to avoid serious modifications:

```
Core classes
Site notices, warnings, errors
Primary Content Columns
Flexible Content Columns
Sub menu - display of tabs in multiple components
```

print.css

This is a style sheet that is used only for printing. It removes unnecessary elements such as menus and search boxes, adjusts any background and font colors as needed to improve readability, and can expose link URLs through generated content (advanced browsers only, e.g. Safari, Firefox).

error.css

This is a style sheet that is used only by the error.php layout. It allows for a more custom styling to error pages such as "404 - Page Not Found".

Internet Explorer

We strongly encourage developers to test their templates in as many browsers and on as many operating systems as possible. Most modern browsers will have little differences in rendering, however, Internet Explorer deserves special mention here.

The most widely used browser, Internet Explorer, is also one of the most lacking in terms of CSS support. Internet Explorer has also, traditionally, handled rendering of block elements, element positioning, and other common tasks a bit differently than many

other browsers. As can be expected, this has led to much controversy and discussion on how best to handle such differences. We strongly recommend designing for and testing your templates in alternate browsers such as Safari, Firefox, Chrome, or Opera and then applying fixes to Internet Explorer afterwards. We recommend the use of conditional comments to apply special Internet Explorer only stylesheets.

Conditional Comments

Conditional comments only work in Internet Explorer on Windows, and are thus excellently suited to give special instructions meant only for Internet Explorer on Windows. They are supported from Internet Explorer 5 onwards, and it is even possible to distinguish between versions of the browser.

Conditional comments work as follows:

```
<!--[if IE 6]>
 Special instructions for IE 6 here
<![endif]-->
```

Their basic structure is the same as an HTML comment (<!-- -->). Therefore all other browsers will see them as normal comments and will ignore them entirely. Internet Explorer, however, recognizes the special syntax and parses the content of the conditional comment as if it were normal page content. As such, they can contain any web content you wish to display only to Internet Explorer. While we're using this feature to load CSS files, it can also be used to load JavaScript or display Internet Explorer specific HTML.

Note: Since conditional comments use the HTML comment structure, they can only be included in HTML, and not in CSS files.

Conditional comments support some variation in syntax. For example, it is possible to target a specific browser version as demonstrated above or target multiple versions such as "all versions of Internet Explorer lower than 7". This can be done with a couple handy operators:

- gt = greater than
- gte = greater than or equal to
- lt = less than
- lte = less than or equal to

```
<!--[if IE]>
 According to the conditional comment this is Internet Explorer
<![endif]-->
```

```
<!--[if IE 5]>
 According to the conditional comment this is Internet Explorer 5
<![endif]-->
<!--[if IE 5.0]>
 According to the conditional comment this is Internet Explorer 5
.0
<![endif]-->
<!--[if IE 5.5]>
 According to the conditional comment this is Internet Explorer 5
.5
<![endif]-->
<!--[if IE 6]>
 According to the conditional comment this is Internet Explorer 6
<![endif]-->
<!--[if IE 7]>
 According to the conditional comment this is Internet Explorer 7
<![endif]-->
<!--[if IE 8]>
 According to the conditional comment this is Internet Explorer 8
<![endif]-->
<!--[if gte IE 5]>
 According to the conditional comment this is Internet Explorer 5
 and up
<![endif]-->
<!--[if lt IE 6]>
 According to the conditional comment this is Internet Explorer l
ower than 6
<![endif]-->
<!--[if lte IE 5.5]>
 According to the conditional comment this is Internet Explorer l
ower or equal to 5.5
<![endif]-->
<!--[if gt IE 6]>
 According to the conditional comment this is Internet Explorer g
reater than 6
<![endif]-->
```

So, to load stylesheets to specific versions of Internet Explorer in our template we do something like the following:

```
<html>
  <head>
    ... other CSS files ...
```

```
    <!--[if IE 7]>
      <link rel="stylesheet" type="text/css" media="screen" href=
"{TemplatePath}/{TemplateName}/css/ie7.css" />
    <![endif]-->
    <!--[if lte IE 6]>
      <link rel="stylesheet" type="text/css" media="screen" href=
"{TemplatePath}/{TemplateName}/css/ie6.css" />
    <![endif]-->
  </head>
  ...
</html>
```

Note: Conditional comments used CSS for should be placed inside the <head> tag of a template *after* all other CSS have been linked for their affects to properly take place.

Loading From An Extension

Components

Often a component will have a style sheet of its own. Pushing CSS to the template from a component is quite easy and involves only two lines of code.

```
HubzeroDocumentAssets::addComponentStylesheet('com_example');
```

First, we load the HubzeroDocumentAssets class. Next we call the static method addComponentStylesheet, passing it the name of the component as the first (and only) argument. This will first check for the presence of the style sheet in the active template's overrides. If found, the path to the overridden style sheet will be added to the array of style sheets the template needs to include in the <head>. If no override is found, the code then checks for the existence of the CSS in the component's directory. Once again, if found, it gets pushed to the template.

Modules

Loading CSS from a module works virtually the same as loading from a component save one minor difference in code. Instead of calling the addComponentStylesheet method, we call the addModuleStylesheet method and pass it the name of the module.

```
HubzeroDocumentAssets::addModuleStylesheet('mod_example');
```

Plugins

Loading CSS from a plugin works similarly to loading from a component or module but instead we call the addPluginStylesheet method and pass it the name of the plugin group **and** the name of the plugin.

```
HubzeroDocumentAssets::addPluginStylesheet('examples', 'test');
```

Plugin CSS must be named the same as the plugin and located within a directory of the same name as the plugin inside the plugin group directory.

```
/plugins
  /examples
    /test
      test.css
    test.php
    test.xml
```

View Helpers (all extensions)

Modules, Component, and plugin views now have helpers for pushing Cascading StyleSheets and JavaScript assets to the document. Each method automatically looks for overrides within the current, active template, taking out the busy work of checking yourself each time assets are added. The method names are short, accept a range of options, and allow for method chaining, all tailored for brevity and ease of use.

The css() method provides a quick and convenient way to attach stylesheets. For components, it accepts two arguments:

1. The name of the stylesheet to be pushed to the document (file extension is optional). If no name is provided, the name of the component or plugin will be used. For instance, if called within a view of the component "com_tags", the system will look for a stylesheet named "tags.css".
2. The name of the extension to look for the stylesheet. For components, this will be the component name (e.g., com_tags). For plugins, this is the name of the plugin folder and requires the third argument of plugin group (type) be passed to the method.

3. *Plugin views only.* The name of the plugin.

Example:

```
<?php
// Push a stylesheet to the document
$this->css()
      ->css('another')  // Extension (.css) is optional
      ->css('tags.css', 'com_tags');  // Load CSS from another co
mponent
?>
... view HTML ...
```

Along with file names, the method also accepts style declarations:

```
<?php
// Push a stylesheet to the document
$this->css('.foo {
 color: #000;
}');
?>
... view HTML ...
```

Similarly, a js() method is available for pushing javascript assets to the document. The arguments accepted are the same as the css() method described above.

```
<?php
// Push some javascript to the document
$this->js()
      ->js('another');
?>
... view HTML ...
```

And, just as the css() method accepts style declarations, the js() method accepts script declarations:

```
<?php
```

```
// Push some javascript to the document
$this->js('
 jQuery(document).ready(function($){
  $("a").on("click", function(e){
   console.log($(this).attr("href"));
  });
 });
');
?>
... view HTML ...
```

Further Help

Resources for learning and sharpening CSS skills:

- CSS Zen Garden
- CSS From The Ground Up
- Guide to Cascading StyleSheets
- CSS School