Prestashop Quick Primer

Understanding Prestashop structure for Designers

Prestashop, like all PHP scripts will load the website using an index.php file. If you prestashop store is located at mystore.com, when a user enters the URL http://www.mystore.com, the webserver will load the index page. It is the information on the index page that determines what happens.

To understand what is going when you load your prestashop shop, we therefore need to understand what the index file does.

INDEX.PHP

The prestashop index.php page (v1.1.0.5) includes the following code

```
<?php
include(dirname(__FILE__).'../../../config/97025AB9-303E-4B9B-A67E-65C6CB061E53.html');
include(dirname(__FILE__).'../../../DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html');

$smarty->assign('HOOK_HOME', Module::hookExec('home'));
$smarty->display(_PS_THEME_DIR_.'../../../index.tpl');

include(dirname(__FILE__).'../../../29BCCA06-66A9-4F89-93B3-CD677D357DB7.html');

?>
```

Firstly the index.php includes the 97025AB9-303E-4B9B-A67E-65C6CB061E53.html file located in the config directory. This file is to do with our configuration and other settings, 97025AB9-303E-4B9B-A67E-65C6CB061E53.html will check if everything is installed and defines our base themes and directory defaults and paths, order statuses and other settings. But we not going to concern ourselves with any further explanations in this designers guide right now, beyond knowing it's included.

However, as a designer, you may need to track errors from time to time so there is one thing I want to make you aware of right now. If you open /config/97025AB9-303E-4B9B-A67E-65C6CB061E53.html you will see a line as follows at the top of the file;

```
@ini_set('display_errors', 'off');
```

If you change this to

```
@ini_set('display_errors', 'on');
```

You have now enabled error reporting. Just remember to turn this back to off in a live installation.

Next, after our config is loaded the index.php file is told to include the DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html file from our root install. Root install means from directly within our prestashop site, not contained in any sub-directories.

This DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html file includes the following code

```
<?php

// P3P Policies (http://www.w3.org/TR/2002/REC-P3P-20020416/#compact_policies)
header('P3P: CP="IDC DSP COR CURa ADMa OUR IND PHY ONL COM STA"');

require_once(dirname(__FILE__).'../../../54381465-2874-4B0B-9796-49D4ABF8285F.html');
```

```
/* CSS */
$css_files[_THEME_CSS_DIR_.'../../../global.css'] = 'all';

/* Hooks are voluntary out the initialize array (need those variables already assigned) */
$smarty->assign(array(
   'HOOK_HEADER' => Module::hookExec('header'),
   'HOOK_LEFT_COLUMN' => Module::hookExec('leftColumn'),
   'HOOK_TOP' => Module::hookExec('top'),
   'static_token' => Tools::getToken(false),
   'token' => Tools::getToken(),
   'priceDisplayPrecision' => _PS_PRICE_DISPLAY_PRECISION_,
   'content_only' => intval(Tools::getValue('content_only'))
));

if(isset($css_files) AND !empty($css_files)) $smarty->assign('css_files', $css_files);
if(isset($js_files) AND !empty($js_files)) $smarty->assign('js_files', $js_files);

/* Display a maintenance page if shop is closed */
if (isset($maintenance) AND (!isset($_SERVER['REMOTE_ADDR']) OR $_SERVER['REMOTE_ADDR'] !=
Configuration::get('PS_MAINTENANCE_IP')))
{
   header('HTTP/1.1 503 temporarily overloaded');
   $smarty->display(_PS_THEME_DIR_.'../../../maintenance.tpl');
   exit;
}

$smarty->display(_PS_THEME_DIR_.'../../../header.tpl');

?>
```

Next we have

```
$smarty->assign('HOOK_HOME', Module::hookExec('home'));
```

This will assign a hook (we will get to hooks in a while, but basically hooks allow the website administrator to specify where they want certain modules to be displayed within your Prestashop shop).

```
$smarty->display(_PS_THEME_DIR_.'../../../index.tpl');
```

This line is saying display the index.tpl file in this position, from the default theme directory. (You define the theme to use in your backoffice>> preferances>>appearance)

And finally we have footer.php

```
include(dirname(__FILE__).'../../../29BCCA06-66A9-4F89-93B3-CD677D357DB7.html');
```

so this line of php is saying include the file 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html from our directory. The 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html file has the following code

```
<?php
```

```
if (isset($smarty))
{
  $smarty->assign(array(
    'HOOK_RIGHT_COLUMN' => Module::hookExec('rightColumn'),
    'HOOK_FOOTER' => Module::hookExec('footer'),
    'content_only' => intval(Tools::getValue('content_only'))));
  $smarty->display(_PS_THEME_DIR_.'../../../footer.tpl');
}

?>
```

(Each file that is included may itself, include PHP requests to include other files of course. But we are only concerned with understanding the principals of what is going on to give you a understanding of how Prestashop works).

Although we now have our configuration setting, Hooks and various php files included, you will notice we don't yet have any HTML. This is where our template (tpl) comes in. Tpl files are the template files that include our HTML and our Smarty code. (We talk more about Smarty in a bit). These tpl files mostly reside in your themes directory but also exist in modules as each module someone writes will need its own template file(tpl) too.

If you now look at the DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html file, we can see the following code is being included with this.

```
$css_files[_THEME_CSS_DIR_.'../../../global.css'] = 'all';
if(isset($css_files) AND !empty($css_files)) $smarty->assign('css_files', $css_files);
if(isset($js_files) AND !empty($js_files)) $smarty->assign('js_files', $js_files);

$smarty->display(_PS_THEME_DIR_.'../../../maintenance.tpl');  -  Will display a maintenance page called maintenance.tpl if your shop is closed */

$smarty->display(_PS_THEME_DIR_.'../../../header.tpl');
```

So our global.css is being called from our theme directory and our required CSS and JavaScript files are included. The actual display HTML (which represents part of our webpage layout) comes from the header.tpl file

Our index.php also calls index.tpl and 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html also calls (or requests) footer.tpl. Again these tpl files determine our sites html layout and exist within out themes folder.

If it sounds a bit complex so far, stick with me, it will get easier,

Ok, so breaking this down, our index.php file includes header.php, index.tpl and footer.tpl from our themes directory. Then DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html is calling header.tpl and 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html is calling footer.tpl.

Thus our main HTML is derived from these three template.tpl files.

Now lets open header.tpl, index.tpl and footer.tpl. This should make a lot more sense to Web designers

Header.tpl

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="{$lang_iso}">
  <head>
    <base href="../../../14FCA9C1-620A-48C1-AC09-CB152DCA08AA.html" />
    <title>{$meta_title|escape:'htmlall':'UTF-8'}</title>
{if isset($meta_description) AND $meta_description}
    <meta name="description" content="{$meta_description|escape:htmlall:'UTF-8'}" />
{/if}
{if isset($meta_keywords) AND $meta_keywords}
    <meta name="keywords" content="{$meta_keywords|escape:htmlall:'UTF-8'}" />
{/if}
    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
    <meta name="generator" content="PrestaShop" />
    <meta name="robots" content="{if isset($nobots)}no{/if}index,follow" />
    <link rel="icon" type="../../../image/vnd.microsoft.icon" href="../../../E4285472-5364-420E-8739-824DE1C8E64C.html" />
    <link rel="shortcut icon" type="image/x-icon" href="../../../E4285472-5364-420E-8739-824DE1C8E64C.html" />
{if isset($css_files)}
  {foreach from=$css_files key=css_uri item=media}
  <link href="../../../1ED416D7-F0D4-4547-B86E-9DCD729724E2.html" rel="stylesheet" type="text/css" media="{$media}" />
  {/foreach}
{/if}
    <script type="text/javascript" src="../../../218ACF9A-85AE-4421-B2A2-36A51DF39156.html"></script>
    <script type="text/javascript">
      var baseDir = '../../../02DED94E-B8A0-4E85-9564-478D39FFF63D.html';
      var static_token = '{$static_token}';
      var token = '{$token}';
      var priceDisplayPrecision = {$priceDisplayPrecision*$currency->decimals};
    </script>
    <script type="text/javascript" src="../../../9FA21342-2022-4E24-8DB1-2F601C4CBF6B.html"></script>
    <script type="text/javascript" src="../../../0AFCEDEE-8C38-464F-A847-19D8378A513F.html"></script>
{if isset($js_files)}
  {foreach from=$js_files item=js_uri}
  <script type="text/javascript" src="../../../B8A0B669-6671-48EA-B0F6-79FD285FCF29.html"></script>
  {/foreach}
{/if}
    {$HOOK_HEADER}
  </head>
```

```
  <body {if $page_name}id="{$page_name|escape:'htmlall':'UTF-8'}"{/if}>
  {if !$content_only}
    <div id="page">

      <!-- Header -->
      <div>
        <h1 id="logo"><a href="../../../02DED94E-B8A0-4E85-9564-478D39FFF63D.html" title="
{$shop_name|escape:'htmlall':'UTF-8'}"><img src="../../../76933532-7FD8-43DC-9519-469A0F8CBBDD.html" alt="
{$shop_name|escape:'htmlall':'UTF-8'}" /></a></h1>
        <div id="header">
          {$HOOK_TOP}
        </div>
      </div>

      <!-- Left -->
      <div id="left_column" class="column">
        {$HOOK_LEFT_COLUMN}
      </div>

      <!-- Center -->
      <div id="center_column">
  {/if}
```

Index.tpl

```
{$HOOK_HOME}
```

Footer.tpl

```
  {if !$content_only}
      </div>

<!-- Right -->
      <div id="right_column" class="column">
        {$HOOK_RIGHT_COLUMN}
      </div>

<!-- Footer -->
      <div id="footer">{$HOOK_FOOTER}</div>
    </div>
  {/if}
  </body>
</html>
```

Now let's View the source for our homepage (With all modules un-installed and a small amount of JavaScript removed from footer so we can focus on the rendered HTML, derived from our key tpl files), here is what we now have

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <base href="http://127.0.0.1/prestashop/" />
  <title>Prestashop demo</title>
  <meta name="description" content="Shop powered by PrestaShop" />
  <meta name="keywords" content="shop, prestashop" />
  <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
  <meta name="generator" content="PrestaShop" />
  <meta name="robots" content="index,follow" />
  <link rel="icon" type="../../../image/vnd.microsoft.icon" href="../../../prestashop/img/favicon.ico" />
  <link rel="shortcut icon" type="image/x-icon" href="../../../prestashop/img/favicon.ico" />
  <link href="../../../prestashop/themes/commotion/css/global.css" rel="stylesheet" type="text/css" media="all" />
    <script type="text/javascript" src="../../../prestashop/js/tools.js"></script>
    <script type="text/javascript">
      var baseDir = '../../../prestashop/Index.html';
      var static_token = '8d6a4d79d983512d770333775a7d8d24';
      var token = '521aacf70fb52a35b4f94e01366c4c64';
      var priceDisplayPrecision = 2;
    </script>
    <script type="text/javascript" src="../../../prestashop/js/jquery/jquery-1.2.6.pack.js"></script>
    <script type="text/javascript" src="../../../prestashop/js/jquery/jquery.easing.1.3.js"></script>
 </head>
  <body id="index">
      <div id="page">
```

```
        <!-- Header -->
        <div>
          <h1 id="logo"><a href="../../../prestashop/Index.html" title="Prestashop demo"><img src="../../../prestashop/img/logo.jpg"
alt="Prestashop demo" /></a></h1>
          <div id="header">

          </div>
        </div>

        <!-- Left -->
        <div id="left_column" class="column">

        </div>

        <!-- Center -->
        <div id="center_column">
            </div>

<!-- Right -->
        <div id="right_column" class="column">

        </div>

<!-- Footer -->
        <div id="footer">
      </div>
      </div>
      </body>
</html>
```

So basically to recap. Index.php loads first, this requests other php pages that are defined in this php file and those php pages then call the tpl files which are together rendered as a webpage.

Notice that index.tpl has just one line: {$HOOK_HOME}. This is beacuse the index file will only show modules assigned to the homepage. Our category and product pages will display other modules.

Also, as you will discover, we have a different body ID on each of these different pages, index.tpl, category.tpl, product.tpl for example so we have the flexibility to style pages differently using just our global.css rules to target the body ID. Where does this come from? If you look at the header.tpl file you will see a line

<body {if $page_name}id="{$page_name|escape:'htmlall':'UTF-8'}"{/if}> and it is this code that gets the page name. So category.tpl will have the <body id="category">

Smarty - A Quick Primer

Smarty is what is called a PHP template engine. Basically you assign variables for use in your templates.The template files (.tpl) then contains the output interspersed with tags that Smarty replaces with the assigned content and displayed as your stores web pages.

Smarty separates your presentation elements (that is, your HTML, CSS, etc.) from your application code. However, it is perfectly fine to use it for tasks such as looping over table row colors or including one template from another which Smarty would still consider presentation logic. The main idea behind Smarty is to keep the template(website) designer role and application programming role separated. So as a designers, once you grasp Smarty, you should be able to modify the Prestashop templates and rebuild them without actually touching the code base while retaining full control of the presentation.

Smarty has some nice features that take advantage of this design principle. One prominant feature of Smarty is variable modifiers. These are used to alter the output of assigned variables from within the template. So if you want the product name to be capitalized out output ina different way, smarty makes this possible. The best way to think about it is that you have the control to alter the final output (what gets displayed) without having to be an experienced php programmer and delve into PHP where you are might break the logic.

There are many benefits as you why Prestashop has chosen Smarty. Two of the most important are;

Security: Templates do not contain PHP code. Therefore, a Prestshop template designer is not unleashed with the full power of PHP, only the subset of functionality made available to them from the prestashop programmers.

Easy to Use and Maintain: Web page designers are not having to deal with PHP code syntax, but instead working with an easy-to-use templating syntax not so much different than plain HTML. The templates are a very close representation of the final output, dramatically shortening the design cycle.

Lets say your using the featured products module. You find the default description displayed is too long at 130 characters. Without the way Prestashop is coded using smarty, you might have had to mess with the PHP. But with the way Prestashop uses the tpl files all you need to do is go to modules/homefeatured, open the tpl file, find this line:

{$product.description_short|strip_tags:htmlall:'UTF-8'|truncate:130}

and simply change 130 to something smaller, say 90 and that it, the description on each product displayed in thsi module is now 90 characters long - no risk of you creating any syntax errors in php. This is a very simple example, but you get the idea.

So Smarty gives more flexibility to designers who are not PHP developers and I recommend that you find the time to get to grips with Smarty if you plan to build many Prestshop sites. Now, if you only want to change the the html and CSS on Prestashop, its not essentail to learn Smarty, but I recommed it.

Where to next: Take a look at http://www.smarty.net/crashcourse.php, then download and install smarty and get familiar.

Understanding Prestashop Themes

Prestashop uses what are called themes to style your your shop. Ideally you will only want to create and edit themes and the themes gloabl.css file and themes images to get your store looking how you want and avoid making changes direct to php files which can be overwritten during future upgrades. The tpl files within your theme are used to render the layout of webpages using Smarty templating engine and HTML. The goal is to have the presentation information held within themes, exclduing the programming logic whcih is in the php files so designers do not have to worry about breaking the code.

Themes contain your template (tpl) files, your CSS rules for the front office style and an image called "../../preview.jpg" which is used by Prestashop within backoffice >> Preferences >>Apprearance to give you a visual image of what each theme looks like.

By default Prestashop ships with one default theme but its easy to add additional themes to your installation.

In addition to tpl files existing with your themes directory, Prestashop also has modules. Modules are the functional blocks such as your category, products, top-sellers etc and allow developers to extend Prestashop in many ways. Modules also need tpl files and these can be found within each modules directory. Therefore, when you need to change the HTML relating to a module, you will need to modify its tpl file. (If if just styling-the glabal.css file is all you would need to work on).

First, I recommend you never work on the default theme. This is because if something is not working right in your shop, you can always change the theme back to the default one and see if the same functional issue exists. The second benefit is that you always have the default theme available if you wish to check a template (tpl file) against changes you may have may on your own theme.

Getting Started

Changing the look of your Theme

For some people, they will be happy with the way the default theme displays the key information blocks and will just want to change the overall style (background images and CSS). In this case, there is no need to do anything with the template (tpl) files.

However, if you need to modify the HTML or even some of the smarty code, you need to work on the tpl files. The best place for us to start is understanding the default HTML and CSS that ships by default with Prestashop and secondly how we edit the tpl files.

Understanding and modifying the default Prestashop theme

Creating new Prestashop themes

Before we start to modify and create our new theme, I always find it's essential to first understand what CSS rules are being applied to the core HTML layouts. I find that creating a document with the core blocks (page, header, left_column, centre, column, right_colum and footer) identified together with the sizes being applied to these columns with the css will help save time throughout the design build and avoid styling issues later on.

To start, we going to make a copy of the default prestashop theme, rename this to "mytheme" and then save the files in the themes directory. I am assuming you already have prestashop 1.1 installed, either locally or on a web server and you are familiar with copying and uploading files.

We will then change preferences to make "mytheme the default", uninstall all the modules and then create the core layout document so we have a viusla for reference. Once we are happy with our main blocks positions and sizes, we will use our pre-designed layout to add the modules we need and style these.

Lets begin…

We are going to create a store design based upon the three column layout but later we will explore changing the templates (tpl files) to produce alternative layouts using 1,2 and 4 columns. Firstly lets duplicate the default theme that comes with prestashop.

Duplicate the default Prestashop theme

Make a copy of the default theme by copying themes/prestshop. Save the folder to your computer and rename this to "mytheme" or whatever name you wish to use. Now copy or upload these renamed folder into the themes directory. You should now have a theme called Prestashop and another called Mytheme in this directory.

Now login to your Prestashop backoffice, go to preferences>>appearance and change the theme to your "mytheme" that you installed and click save.

Uninstalling modules

I like to have everything uninstalled when I begin and then work from the outside in, starting with the key HTML blocks (such as our columns). I find this makes it much easier to check everything is working correctly across all popular browsers, is validated and the diemsions add up. Once we know this is correct we can lock down this part of our rules before moving onto styling the content itself. If we havea problem later, say with a column dropping down, we know its something we have added to the design that has caused this and not our column CSS.

Log into your Prestashop backoffice. Then go to modules. Now click the uninstall button for all modules that are currently installed.

Create a layout reference document.

We are going to base our layout reference document on the "mytheme" HTML of our site that comes from our core tpl files. As previously discussed, this is made up primarily from the header.tpl and footer.tpl files.

When combined, the header and footer.tpl files contain the following HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="{$lang_iso}">
  <head>
    <base href="../../../14FCA9C1-620A-48C1-AC09-CB152DCA08AA.html" />
    <title>{$meta_title|escape:'htmlall':'UTF-8'}</title>
{if isset($meta_description) AND $meta_description}
    <meta name="description" content="{$meta_description|escape:htmlall:'UTF-8'}" />
{/if}
{if isset($meta_keywords) AND $meta_keywords}
    <meta name="keywords" content="{$meta_keywords|escape:htmlall:'UTF-8'}" />
{/if}
    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
    <meta name="generator" content="PrestaShop" />
    <meta name="robots" content="{if isset($nobots)}no{/if}index,follow" />
    <link rel="icon" type="../../../image/vnd.microsoft.icon" href="../../../E4285472-5364-420E-8739-824DE1C8E64C.html" />
    <link rel="shortcut icon" type="image/x-icon" href="../../../E4285472-5364-420E-8739-824DE1C8E64C.html" />
{if isset($css_files)}
  {foreach from=$css_files key=css_uri item=media}
  <link href="../../../1ED416D7-F0D4-4547-B86E-9DCD729724E2.html" rel="stylesheet" type="text/css" media="{$media}" />
  {/foreach}
{/if}
    <script type="text/javascript" src="../../../218ACF9A-85AE-4421-B2A2-36A51DF39156.html"></script>
    <script type="text/javascript">
      var baseDir = '../../../02DED94E-B8A0-4E85-9564-478D39FFF63D.html';
      var static_token = '{$static_token}';
      var token = '{$token}';
      var priceDisplayPrecision = {$priceDisplayPrecision*$currency->decimals};
    </script>
    <script type="text/javascript" src="../../../9FA21342-2022-4E24-8DB1-2F601C4CBF6B.html"></script>
    <script type="text/javascript" src="../../../0AFCEDEE-8C38-464F-A847-19D8378A513F.html"></script>
{if isset($js_files)}
  {foreach from=$js_files item=js_uri}
  <script type="text/javascript" src="../../../B8A0B669-6671-48EA-B0F6-79FD285FCF29.html"></script>
  {/foreach}
{/if}
    {$HOOK_HEADER}
  </head>
```

```
  <body {if $page_name}id="{$page_name|escape:'htmlall':'UTF-8'}"{/if}>
  {if !$content_only}
    <div id="page">

      <!-- Header -->
      <div>
        <h1 id="logo"><a href="../../../02DED94E-B8A0-4E85-9564-478D39FFF63D.html" title="
{$shop_name|escape:'htmlall':'UTF-8'}"><img src="../../../76933532-7FD8-43DC-9519-469A0F8CBBDD.html" alt="
{$shop_name|escape:'htmlall':'UTF-8'}" /></a></h1>
        <div id="header">
          {$HOOK_TOP}
        </div>
      </div>

      <!-- Left -->
      <div id="left_column" class="column">
        {$HOOK_LEFT_COLUMN}
      </div>

      <!-- Center -->
      <div id="center_column">
  {/if}
  {if !$content_only}
      </div>

<!-- Right -->
      <div id="right_column" class="column">
        {$HOOK_RIGHT_COLUMN}
      </div>

<!-- Footer -->
      <div id="footer">{$HOOK_FOOTER}</div>
    </div>
  {/if}
  </body>
</html>
```

These two tpl files also contain our hooks. For example in the right_column div is the following code {$HOOK_RIGHT_COLUMN}

When you add modules in the Prestashop backoffice you specify where these modules should be placed using positions. So if you "add the cart module to the right column using positions", the cart will be placed where {$HOOK_RIGHT_COLUMN} exists in our tpl file which of course is the right_column. In fact, if you wanted to test this, you could move the code {$HOOK_RIGHT_COLUMN} into the footer div and resave your footer.tpl file. You will of course need to re-insall a module such as cart. Once installed, if you refesh you website home page in your browser, you would see that modules that had been placed into {$HOOK_RIGHT_COLUMN} would now be shown in the footer (subject to what CSS styling rules apply of course).

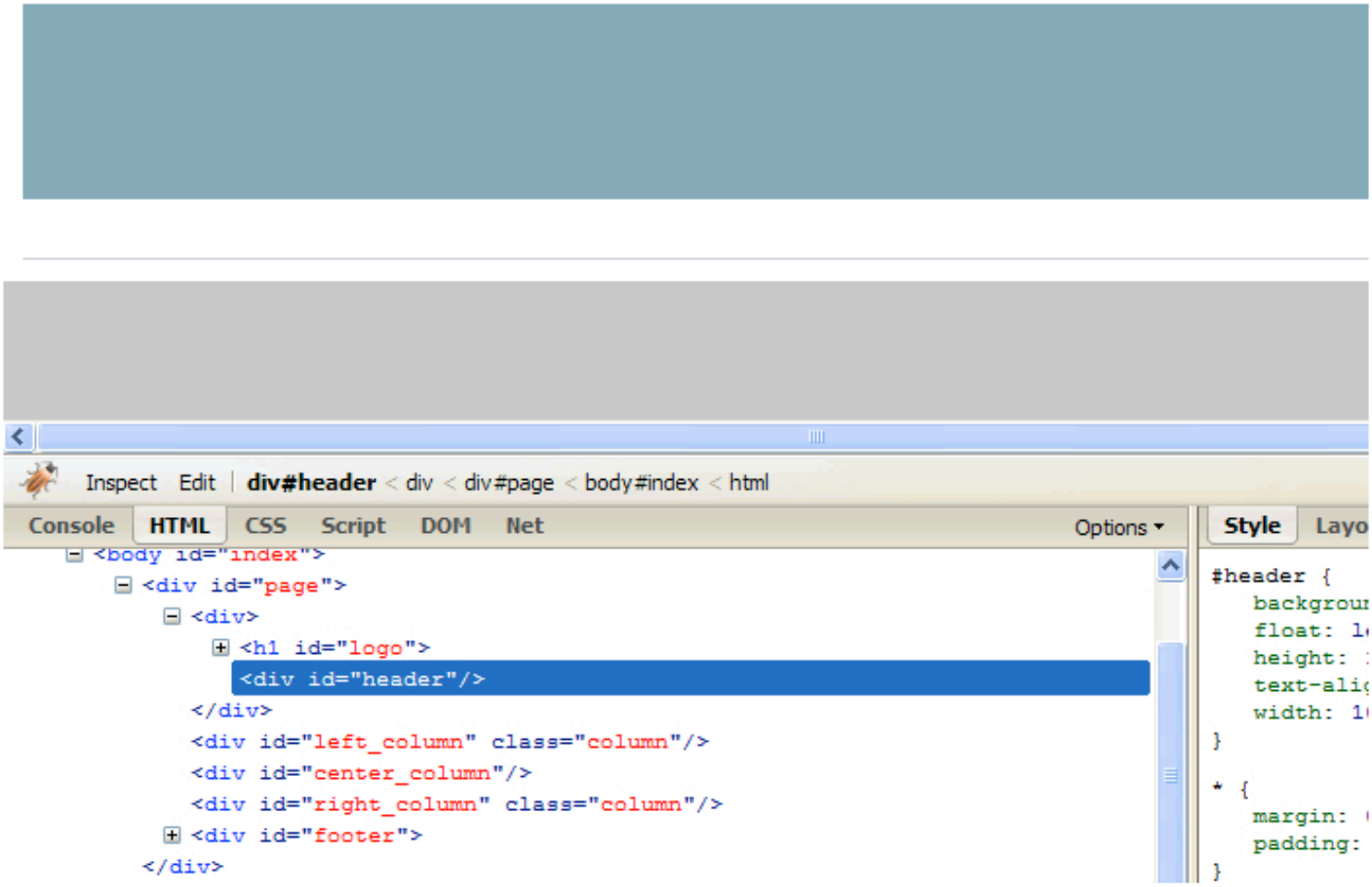
You may recall that our sites index.php file in the root installation also included $smarty->assign('HOOK_HOME', Module::hookExec('home')); simply stated this is saying if the page is the home page, display all modules that have been assigned to the home page which is {$HOOK_HOME}. Hopefully you get the idea, the hooks code in these tpl files determine where modules end up being displayed in your website.

Now lets strip the above HTML from our two tpl files above down to the basics we are interested in right now.

```
<html>
 <head></head>
   <body>
     <div id="page">the wrapper containing all your site information<div>
   <div>
       <h1 id="logo">logo...</h1>
       <div id="header">header content...</div>
   </div>
  <div id="left_column" class="column">left column content...</div>
  <div id="center_column">centre content...</div>
  <div id="right_column" class="column">right column content...</div>
  <div id="footer">footer content...<div>
     </body>
</html>
```

Having identified the key blocks (our divs as shown above) we can now identify the CSS rules that determine how the html blocks will be styled. During this exercise you will see that we are going to work from the main HTML blocks inwards. We are also going to stick with a 3 column layout. (but had we wanted to start with no right column for example, we could remove the #right_column from the footer.tpl file - we will get to complete new layouts in the future).

Firstly, the easiest way to identify the CSS rules that determine how these main div blocks are displayed is to load up your new prestashop home page in the firefox browser using firebug (a firefox add-on). You can install Firebug by choosing Tools>>Add-ons in Friefox and once installed, run Firebug by selecting tools>>firebug>>open firebug.



Once open, Firebug allows you to hover over the html elements, understand the DOM structure and identify what CSS rules are being applied to the selected HTML element - Firebug will even tell you

Within this right column under the tab "style", as we click on the HTML element shown in the left column of firebug, it will show us what CSS rules are being applied to the HTML tag. We can use this right column to tell us what rules are being applied to each HTML tag to click, which line in your CSS file the rule is coming from and even edit the CSS rules on the fly to see a "Live view" of the changes rendered as a webpage.

To out this into practice, click on div id=page (you need to select the actual div element itself). Now in the right window you should be able to see that #page has a width of 980px. Make a note and move through the other key divs, noting the sizes applied and if there are any other rules that affect our core HTML column dimensions such as padding. If you theme is the same as mine, you should have the following settings, although if it varies that it fine.

My settings

#page is 980px
#logo is 29%
#header is 71%
#left_column is 190px with 15px of padding to the right (note that there is also a class applied which will apply additional rules)
#centre_column has a width of 556px
#right_column is 190px with 15px of padding to the left
#footer has no width defined but a top-border of 1px

there is also a CSS rule that applies left floats; (global.css (line 198)

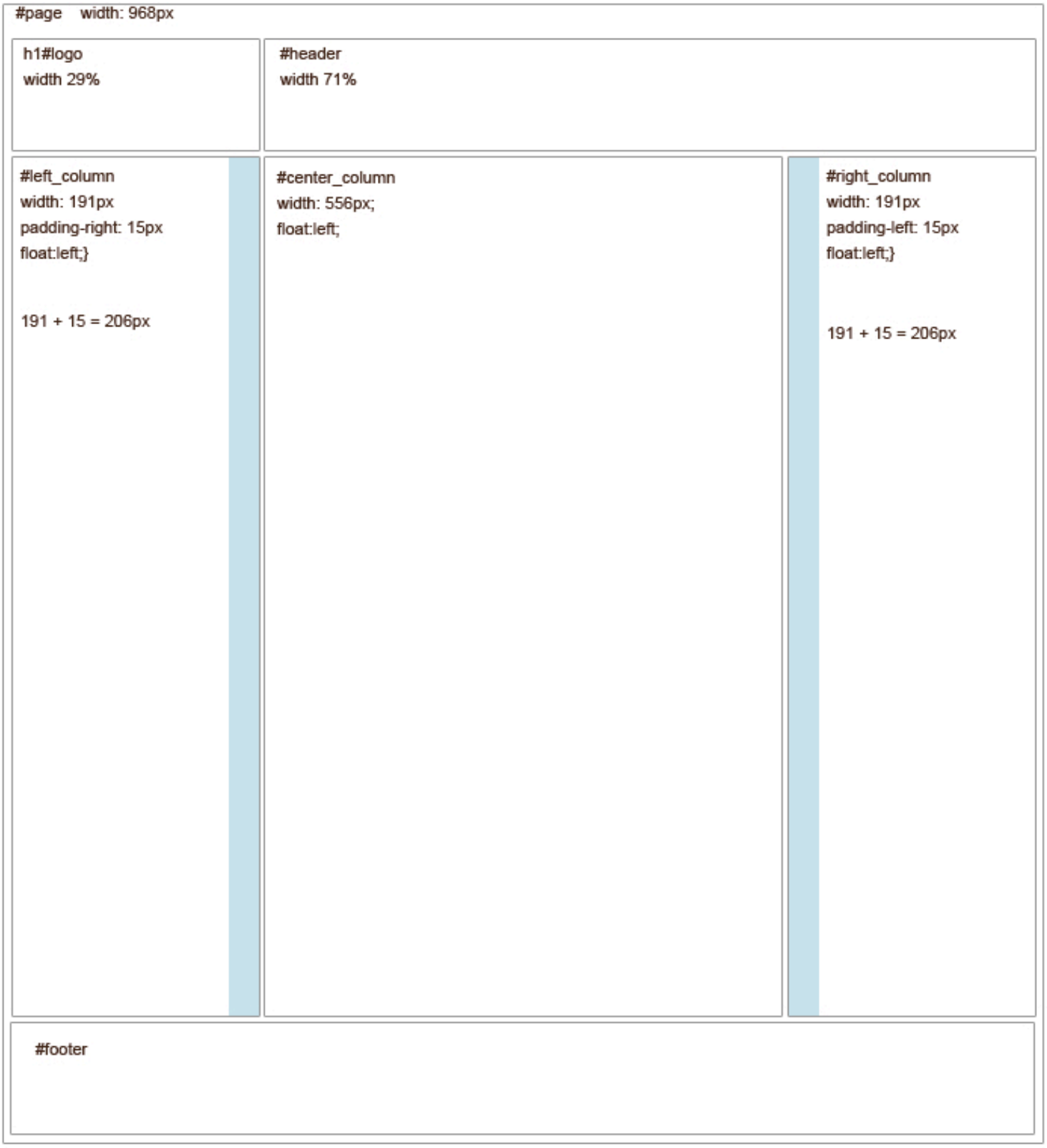
{#left_column, #center_column, #right_column
float:left;}

So we know each main column is floated left and the footer is set to clear these floats - a pretty typical set-up.

Now that I have established teh container (in our case #page) is 980px wide I can calculate the widths of the our three columns inside #page. The left and right column are 190px + 15px padding (205px wide) and the centre column is 556px. So our columns add up to a total of 966px and are placed inside #page which is 980px, a difference of 14px. Firstly, I have no idea why there should be a difference of 14px in the css between #pahe and its 3 main columns, therfore I prefer to change it to be correct for what I want rather than have pointless space. So I could already change the overall page width to 966px knowing this would fit by columns precisely side by side without forcinging the right columns to drop down.

(Tip: Although our left and right columns are 190px wide I can also already see in the CSS (well my themes global CSS file anyway), the block CSS, advertising_block and payment logos are all set to 191px wide, 1px wider than the columns they are going into.) As its only 1px you are unlikely to spot the error in the rendered web page but lets change your left and right column css to 191px (if your are also seeing 191px) and the #page width to 968px at this stage. That way we know all the dimensions are correct. Hopefully you can already start to appriciate why I like to work this way.

I recommend either using graph paper or whatever computer program you prefer, that you draw out these blocks with your required dimensions shown. This will form part of your working layout sheet which, as your design gets more complex, will prove invaluable. Below is my simple layout graphic using the revised dimensions that will be applied to my new globalNew CSS very soon. Padding is highlighted in light blue.



As we start to add to our site, we will be updating our simple block graphic above so we can look in detail at whats going on.

Modifying our new theme - Page 2

Having made a copy of the themes directory we are now going to set up a new cascading stylesheet(css) file within our theme directory

Why am I doing this?

I don't want to work on the default global css file, its very large and has a lot of CSS rules so that will make it confusing and complex to edit. By taking this approach of adding rules to my new file as I go along I can make changes on a gradual basis and easily test, knowing that due to cascade order -because our gloablNew.css file will come after the default global.css file- it will override the rules in this file.

If we find issues as we develop our css rules, at any point I can remove global.css or my new globalNew.css file completely and just look at the styling based on either style sheet. I also have the benefit of being able to compare CSS files should we find a rendering problem with changes we have made that is not apparent on the default themes css file. However because I still have the css rules from global.css accesses by default most of the time I can ensure I will still see all modules displayed using the default Prestashop style sheet making development much easier

Overall, as I say, there does seem to be a lot of CSS rules in Prestashop. It's a shame that the CSS for a module could not be added only when that Module is used, this would help reduce out file size and complexity. However this might mean linking each module to its own CSS file and with some rules having to exist in the global file in any event, this would present other challenges.If you know you will never use a given module you can of course remove the css from our gloablNew file as needed

Adding a new Style sheet to our theme

Firstly make copy of global.css and rename copy to globalNew.css. Then remove all rules from this file except those for our main blocks with the dimensions we have dicussed . (If you want to avoid the time in doing this, you can also download this globalNew.css file from here).

Now upload this file to your new themes/css directory.

Next we will add the link to our new css file. To do this open Header.php which is located in your Prestashop root directory.

Copy the line $css_files[_THEME_CSS_DIR_.'../../../../global.css'] = 'all'; and paste this directly below, then change global.css to gloablNew.css making sure this line comes after global.css. You code should look like this;

```
/* CSS - second  link to gloablNew added by DE*/
$css_files[_THEME_CSS_DIR_.'../../../../global.css'] = 'all';
$css_files[_THEME_CSS_DIR_.'../../../../globalNew.css'] = 'all';
```

Save the file.

Many designers like to use different stylesheets for typography, layout and branding. I am no exception, but for this exercise we are simply going to use our one extra stylesheet.

The first thing I want to do is apply my dimensions for my main columns and divs that are now part of our layout document that we discussed in the previous page. Our left and right columns will be 191px wide , and the #page div will be 968px wide. We will also add a white background to #page

Open globalNew.css and add the following rules under /* global layout */ around line 80;

```
/* global layout */

#page {
width: 968px; background: #fff;
   margin: 0 auto 2px auto;
   text-align:left; padding: 0 5px;}

#left_column, #center_column, #right_column {
   float:left;}

#left_column {
   clear:left;
   width:191px;
   padding-right: 15px;}

#center_column {width: 556px;}

#right_column {
   width: 191px;
   padding-left: 15px;
}
```

Next I want to change the default background color used on BODY so I can see visually where teh #page finishes during the build as well as the font to font-family: Arial, Helvetica, Sans-Serif; and the size used to 12pixels. Personally I like a slightly larger default font size. Leading will also be set at 150% or 18pixels. Later we will also use a default bottom margin of 18pixels so we know the exact height each line will occupy.

Pixels not ems!

Some designers will scream "don't use pixels" you cannot resize type. Well, for the last 4 years I used ems, but now with browsers supporting page zoom and only Internet Explorer not supporting text resizing in pixels (may may be resolved in IE8) I feel going back to pixels is perfectly OK. Actually, I think most people who need to see the page bigger will prefer page zooming anyway. It is extremely hard, especially with eCommerce sites to keep a design looking right when a visitor just increases text sizes alone. (If we are targeting people who need text sizing –we can of course add a style sheet switcher, where we can still check the layout at increased sizes and adjust that specific style sheet to deal with any layout issues

With CSS, remember that leading is added to the top and bottom of your type. This means 12px type with 18px line-height (leading) will add 3px to the top and bottom (3 + 12 + 3). By understanding our default type height we can be more precise with images, know that if we want 4 line of type to sit exactly next to an image we can make the image 72px height (4 x 18px). To account for the 3px above the first line of text we can also add 3px of top margin. This way we avoid type sitting partly below the bottom of the image but still next to it

Having changes by Default type to 12px, I recheck the layout. Immediately I can see the product page now has a problem, the description box (div) to the right is now below the main image. Why has this happened? What is happening is that someone has set the margin in ems. Our increase in type size has increased this margin which now means we would have to increase the #centre_column and #page by 2px to fix this. But using ems for margins in combination with fixed widths is a very bad idea and with floats we are just asking for trouble. We will therefore change this margin to pixels in our globalNew.css. (Note: If you are using my downloaded globalnew.css file you will find I have already made the changes to the margin. If not you need to complete the following;

#centre_column has a width of 556px. Inside this we have

```
#primary_block #pb-right-column
{float:left;
width:310px;
}
```

and inside this we have

```
#primary_block #image-block {
border:1px solid #D0D1D5;
height:300px;
width:300px;
}
```

To the right, but also inside we have

```
#primary_block #pb-left-column {
float:left;
margin-left:1.1em;
width:233px;
}
```

So if our primary block is 310px and the #primary_block #pb-left-column is 233px, that's 543px. Our margin-left is taking our default text size of 12px and multiplying this by 1.1, giving 13.2. If we add these figures together we have 556.2px. As the centre_column is 556px wide it no longer fits so float down below the product image. So we need a margin_left of 13px to fit precisely (556 - 543). But we will set this to #margin_left 12px giving us 1px of breathing space.

Change (as indicated in bold) the margin from 1.1em to 12px

```
#primary_block #pb-left-column {
float:left;
margin-left:12px;
width:233px;
}
```

Having saved the file, now we test the page in the browser so we can see everything looks fine on the products page. (PS.I can see that there is a seperate problem with search box in the header-our smaller #page width means this no longer fits and has dropped down-but we will deal with this later, when we start to style the header)

Now, before we add these css rules for the products page into our globalNew.css file, we will take a look at the borders being applied. We have borders applied in two places and I only want to see this around the image so will remove this from #primary_block #image-block and add this to #primary_block #image-block img in our globalNew.css. (There may be a reason why its in two places – but we will cross that bridge when we come to it)

OK, around line 724 under the heading /* product.tpl */ add the following rules back to your globalNew.css file and then save.

```
#primary_block #pb-left-column {
float:left;
margin-left:12px;
width:233px;
}

#primary_block #image-block {
height:300px;
width:300px;
}

#primary_block #image-block img { border:1px solid #D0D1D5;}
```

(Remember that global.css will still apply rules where there is nothing in our globalNew.css file to override these)

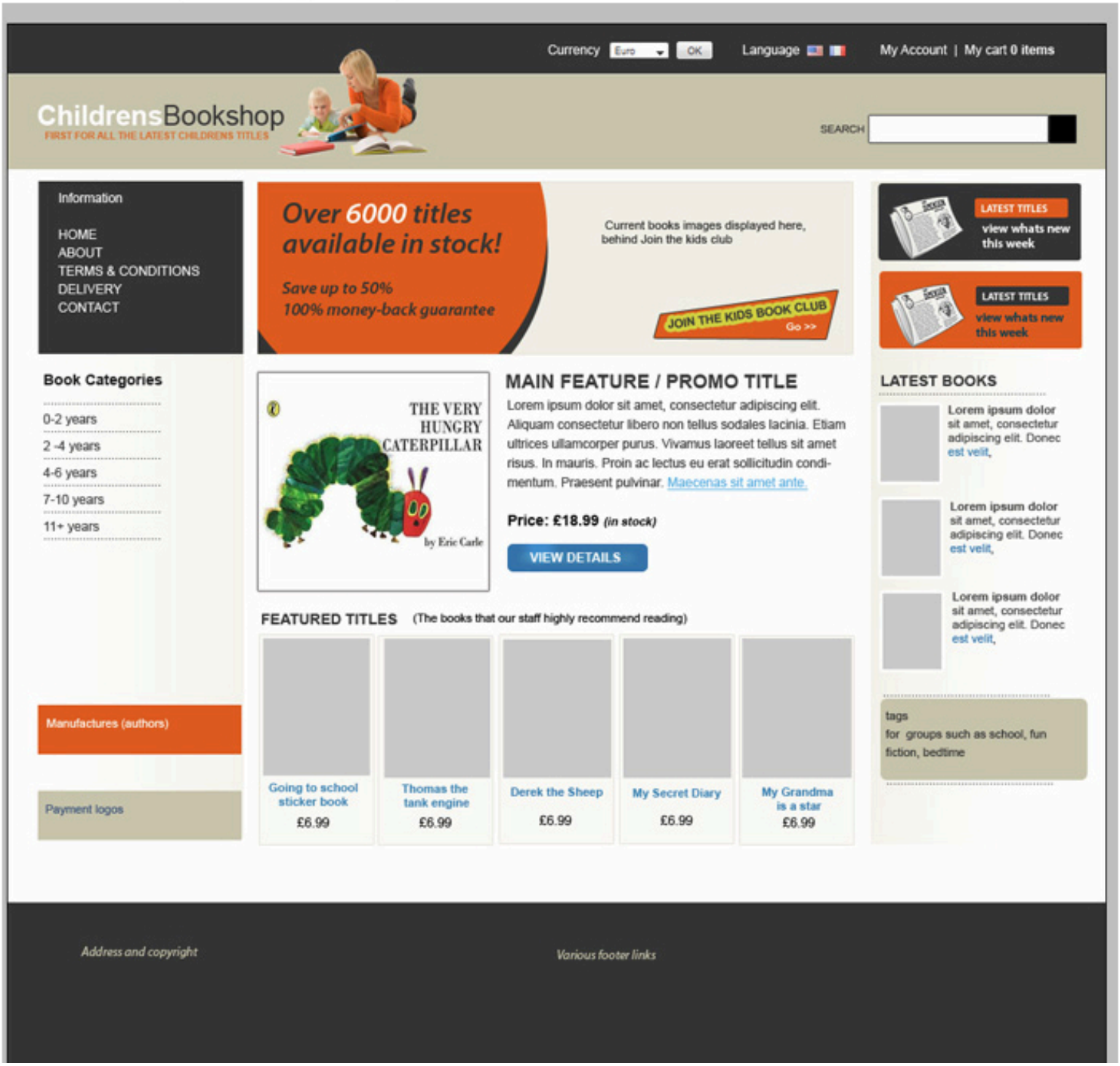
Next we are going to look at the graphic design for our eCommerce store.

Modifying the default theme - Page 3

Firstly, when is comes to building your own theme you will find its a good idea to think about what you will need to achieve, who your audience will be, what they will want to do and therefore what information should be on the site and why, before you start the design. Also think about which modules you will want to use and where you plan to place these modules within your proposed design and why. Good research and preparation and resisting the temptation to jump straight in at the deep end will pay you dividends in the long run.

Now, turning to this exercise, our new shop will be based on a thriving bookstore (not very exciting to some people I know, but it will more than serve our purposes). I have created a pretty basic design as the basis for our new Prestashop bookstore and also given some thought to what modules will be placed where on the home page. Just remember that the goal is simply to take you through the process from how a design becomes a finished Prestashop theme. This is not a design lesson and we are not trying to win any designs awards. Indeed, we are likely to stick with gray boxes for most our book cover images in the graphic below as this will serve our purposes.

Our Prestashop home page design



Now that we have our design agreed, we will start work on incorporating this into the "Mytheme" theme directory we have created. We will work from the outside in (with the approach we are taking here) and use much of the existing HTML and CSS rules already built into Prestashop. However, like many designers, I would have preferred to scrap most of the html information in the tpl files and then re-write these with more efficient HTML tags and CSS rules which would bring a lot more order and make most of the CSS re-suable so new module developes for example, do not need to re-create new ID's, and classes every time which would be a lot more efficient.)

Not only are we going to style the tpl files for our theme- including the modules tpl files we will be using- we will also be creating new hooks so you can see first hand exactly how to do this. Later, we will also delve into both Smarty and JQuery as we'll be making some changes here to meet our specific needs for our bookshop

Until next time…

Understanding Prestashop

Introduction

To understand how Prestashop fully works, you need a understanding of PHP and Smarty. If you are a designer with limited PHP skills, dont panic. You will still find this section important if you wish to modify your store beyond simply changing your theme template files (.tpl) and the global css files. But of course, for many people, the theme may be all you wish to change.

Prestashop uses Smary templating, and you will be surpised how quickly you can learn Smarty without having to be a PHP/MySQL programmer. This will extend your HTMl/CSS skills and provide you with a greater level of flexibility.

lets get started...

Core (Root) PHP files

Within Prestashop, within your root are the folders and the core PHP files.

404.php
address.php
addresses.php
authentication.php
best-sales.php
cart.php
category.php
CHANGELOG
cms.php
contact-form.php
discount.php
footer.php
get-file.php
header.php
history.php
htaccess.txt
indentity.php
images.inc.php
index.php
init.php
manufacturer.php
my-account.php
new-products.php
order/php
order-confirmation.php
order-detail.php
order-follow.php
order-return.php
order-slip.php
pagination.php
password.php
pdf-invoice.php
pdf-order-return.php
pdf-order-slip.php
prices-drop.php
product.php
product-sort.php
README
robots.txt
search.php
sitemap.php
statistics.php
supplier.php
zoom.php

if we now look at the 96F56C94-4F0D-41F9-8803-DD5BA66C8484.html file, it contains the following code

```
<?php
include(dirname(__FILE__).'../../config/97025AB9-303E-4B9B-A67E-65C6CB061E53.html');
include(dirname(__FILE__).'../../DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html');

$smarty->assign('HOOK_HOME', Module::hookExec('home'));
$smarty->display(_PS_THEME_DIR_.'../../index.tpl');

include(dirname(__FILE__).'../../29BCCA06-66A9-4F89-93B3-CD677D357DB7.html');
?>
```

Bascially this is the main index page when your prestashop is loaded and it tells Prestashop to included the configuration file "../../97025AB9-303E-4B9B-A67E-65C6CB061E53.html", the DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html file, the Home Hook, where the theme index template is that styles this homepage and includes the 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html file

if we now look at the 29BCCA06-66A9-4F89-93B3-CD677D357DB7.html file being called into the index file, it contains the following code

```
<?php

if (isset($smarty))
{
  $smarty->assign(array(
    'HOOK_RIGHT_COLUMN' => Module::hookExec('rightColumn'),
    'HOOK_FOOTER' => Module::hookExec('footer'),
    'content_only' => intval(Tools::getValue('content_only'))));
  $smarty->display(_PS_THEME_DIR_.'../../footer.tpl');
}

?>
```

This file assigns the hooks for the right_column and footer (Hooks are used to allow you to easily move modules form one block such a the left_column to the right_column and covered here) and calls the template file footer.tpl from your default theme

If you looked at the DAEAE260-EE55-41A6-B49C-ACD0EDC7AEBC.html file also being called from the 96F56C94-4F0D-41F9-8803-DD5BA66C8484.html file you will see this includes the header info, the hooks for the header, Top and left-Columns and uses the header.tpl file from your theme to style the HTML for the header, top and left_column.

So the tpl files in your themes contain the smarty code (more in this in a moment). Within your themes is a CSS/global.css file and it is this file that control most of your FrontOffice (what your web visitor sees) store layout and design. This gobal.css file is called from the 96F56C94-4F0D-41F9-8803-DD5BA66C8484.html file in your site root (not the tpl files) with the following code;

```
/* CSS */
$css_files[_THEME_CSS_DIR_.'../../global.css'] = 'all';
```

Here are the Hook Table's:

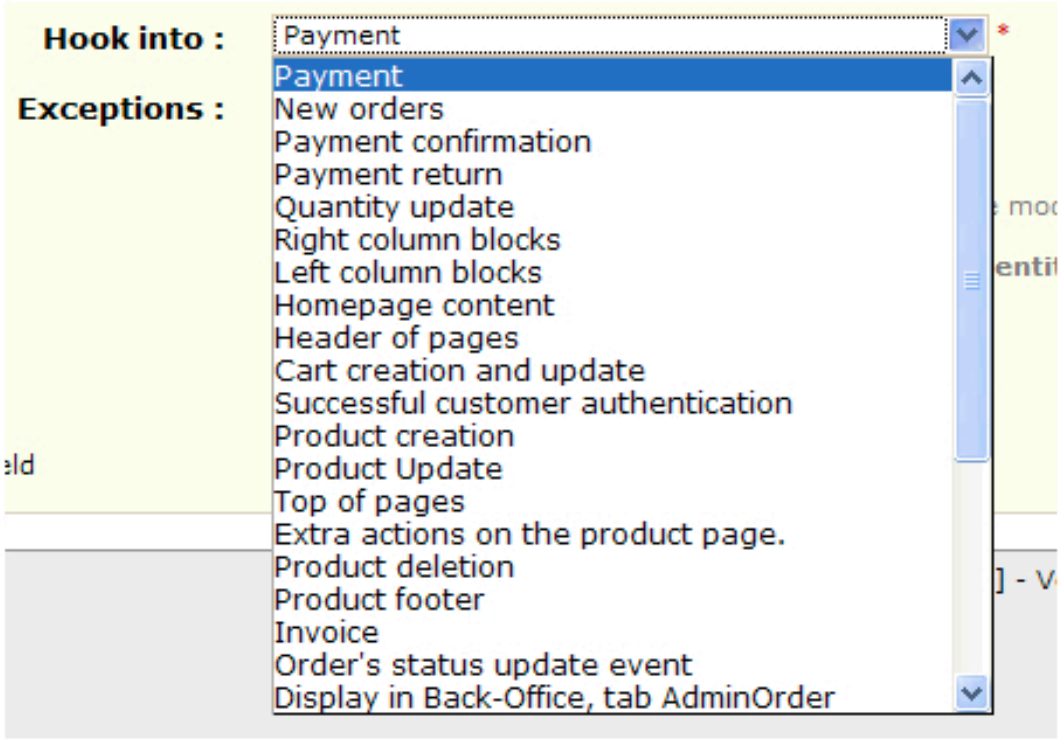
ps_hook

1. payment
2. newOrder
3. paymentConfirm
4. paymentReturn
5. updateQuantity (Quantity is updated only when the customer effectively places his order.)
6. rightColumn (Add blocks to rightColumn)
7. leftColumn (Add blocks to leftColumn)
8. home (Add blocks to Homepage content mainpage)
9. header (Add blocks to header)
10. cart (Cart creation and update)
11. authentication
12. addproduct
13. updateproduct
14. top (A hook which allow you to add modules to the top of each pages.)
15. extraRight (Extra actions on the product page (right column)
16. deleteproduct
17. productfooter (Add new blocks under the product description on the page)
18. invoice (Add blocks to invoice (order))
19. updateOrderStatus (Order's status update event - Launch modules when the order's status of an order changes.)
20. adminOrder (Launch modules when the tab AdminOrder is displayed in the back Office.)
21. footer (Add blocks to footer)
22. PDFInvoice (Allows the display of extra information on the PDF invoice)
23. adminCustomers (Launch modules when the tab AdminCustomers is displayed on back-office.)
24. orderConfirmation (Called on order confirmation page)
25. createAccount (Called when a new customer creates an account successfully)
26. customerAccount (Will display in the Customer account page in the front office)
27. orderSlip (Called when a quantity of one product is changed in an order or order slip created)
28. productTab
29. productTabContent
30. shoppingCart (Display some specific information on the bottom of the shopping cart page)

In short, When you "Hook into" you are saying use this module in this place in my website. So if you were installing categories into the left_column you would via backOffice>>Modules>Positions>>Transplant a module, chosse the Categories block in modules and hook into Left Column Blocks. remeber however that not all modules can be hooked into all places.

Understanding Hooks for Modules in backoffice

When you use the "transplant a module" features in the backoffice, sometimes it can be difficult to know what hooks exist and what they relate to in the drop down list (Screenshot below). Hopefully you will see how this drop down list relates to the Hook table above.



Ps_hook_module

This table stores the module ID, the Hook ID and its position

hook_module_exceptions

Exceptions allow you to specify the pages where you do not wish this module to appear. When you make exceptions and save thise in the backOffice, they are written to this database table. These exception pages relate to your core Prestashop PHP pages, such as products.php or order.php so if you has stated to put (hook into) categories in the elft-Column, the categories would appear in the left_column on all pages except products because you have made this an exception.