# Magento for Designers: Part 5

Magento is a stunningly powerful e-commerce platform. In this miniseries, we’ll learn how to get started with the platform, getting to know the terminologies, setting up a store and all related aspects of it and finally learn how to customize it to make it our very own.

In this fifth part, we’ll be building one of the main views of our store – the product information page. Excited? Let’s get started!

## The Full Series

* [Part 1: Installation and Setup](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-i/)
* [Part 2: Products, Taxes, Categories, Payment Gateways, etc.](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-2/)
* [Part 3: Theming](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-3/)
* [Part 4: Building the Theme](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-4/)
* [Part 5: Building the Product Information Page](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

## A Quick Recap

In the last part, we laid down the framework for the theme by building the repeating portions of the theme and essentially defining the general layout of the theme.

We also took a practical look at how theming in Magento works by going through how the layout is constructed, how the blocks work and how all the different pieces of the puzzle fit in together.

## What are We Building Today?

Today, we’ll build the individual product view page. Just like before, the source files, both the front end and back end, are included. You can use it to see how the page looks but outside of that, the theme should look broken since, you know, we haven’t touched the rest of the views yet. So till we build the rest, try to not venture out of our sandbox.

## Goals for the Page

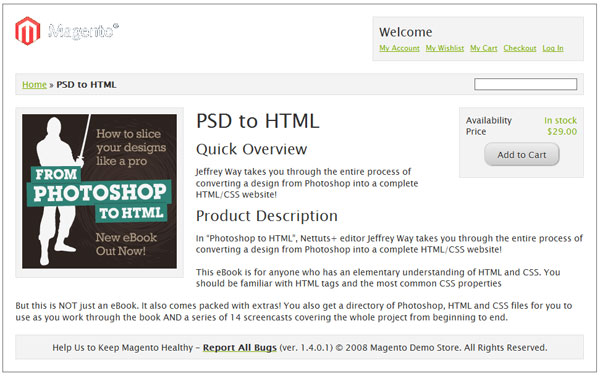
Our goal for this specific page is relatively simple. We want a no-nonsense page that does only the essentials. Thus, I’ve drawn up a short list of elements that I think it needs:

* The product’s title: Fairly obvious
* Ability to show an image of the product
* A quick and full overview
* Availability and price of the product and finally
* An add to cart button to place it in the cart

That’s it. I want to keep it as simple as possible and thus opted to not over do it. Once you’ve learnt the general principles, feel free to add as many whizbang features as possible.

## The Basic Look

The page basically has to look like so:



## Step 1 – The HTML

We’ll first look at the HTML for the content part alone. I’m assuming you’re fairly fluent in HTML and CSS so I’ll skip to the fairly important parts.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <div id="content" class="product">
2. <div id="main-product-image"><img src="images/primg.gif" /></div>
4. <div id="product-details">
5. <div id="product-availability">Availability <span class="available">In stock</span></div>
6. <div id="product-price">Price <span>$29.00</span></div>
7. <a class="button" href="#">Add to cart</a>
8. </div>
10. <h1>Photoshop to HTML</h1>
12. <h2>Quick Overview</h2>
13. <p class="quick-overview">
14. </p>
16. <h2>Product Description</h2>
17. <div class="product-description">
19. </div>

First up, notice that I’ve wrapped the product image with a div to make it easier to add features in the future. Suppose you want to add a quick caption to the displayed image in the future, this way will let us add this a lot more quicker.

We have another div element holding the availability and pricing information along with the add to cart button. We’ll be sprucing it up with a little CSS3 in a bit.

The rest of the HTML is very straightforward. The title resides in a h1 section while the individual section headings take up h2. The quick overview is nested in a paragraph while the full overview takes up a div.

We can move on to the styling part now.

## Step 2 – The CSS

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. /\* Product page \*/
3. #main-product-image {
4. margin: 0 20px 10px 0;
5. padding: 10px;
6. float: left;
7. border: 1px solid #E1E1E1;
8. background: #F3F3F3;
9. }
11. #product-details {
12. width: 180px;
13. padding: 10px;
14. float: rightright;
15. border: 1px solid #E1E1E1;
16. background: #F3F3F3;
17. margin: 0 0 0 20px;
18. }
20. #product-availability span.available, #product-price span {
21. color: #7db000;
22. float: rightright;
23. }
25. .button {
26. margin: 10px auto;
27. display: block;
28. width: 140px;
29. padding: 5px 10px;
30. text-align: center;
31. text-decoration: none;
32. color: #555;
33. border: 1px solid #ccc;
34. font-size: 18px;
35. background: #ddd;
36. border-radius: 12px;
37. -webkit-border-radius: 12px;
38. -moz-border-radius: 12px;
39. box-shadow: 1px 1px 2px rgba(0,0,0,.5);
40. -webkit-box-shadow: 1px 1px 2px rgba(0,0,0,.5);
41. -moz-box-shadow: 1px 1px 2px rgba(0,0,0,.5);
42. text-shadow: #fff 0px 1px 1px;
43. background: -webkit-gradient(linear, left top, left bottombottom, from(#eeeeee), to(#cccccc));
44. background: -moz-linear-gradient(top,  #eeeeee,  #cccccc);
46. }
48. .button:hover {
49. background: #014464;
50. background: -webkit-gradient(linear, left top, left bottombottom, from(#cccccc), to(#999999));
51. background: -moz-linear-gradient(top,  #cccccc,  #999999);
52. color: #000;
53. }
55. .button:active {
56. -moz-box-shadow: 0 2px 6px black;
57. -webkit-box-shadow: 0 2px 6px black;
58. }

Nothing fancy here. Very basic CSS to place the elements in position.

I’ve also used a bit of CSS3 to make the buttons a little better looking.

## Step 3 – Creating our catalog.xml File

As I mentioned in the earlier part, each module gets its own XML file to dictate what items to include and the general layout of the page. The page we’re building today relies on a file called catalog.xml to define its contents and structure.

This file should be present in the layout folder so let’s create an XML file and name it catalog.

The complete file for today looks like so. I’ll explain each bit part by part below.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?xml version="1.0"?>
2. <layout version="0.1.0">
3. <catalog\_product\_view translate="label">
4. <label>Catalog Product View (Any)</label>
5. <!-- Mage\_Catalog -->
6. <reference name="root">
7. <action method="setTemplate"><template>page/1column.phtml</template></action>
8. </reference>
9. <reference name="head">
10. <action method="addCss"><stylesheet>css/product.css</stylesheet></action>
11. </reference>
12. <reference name="content">
13. <block type="catalog/product\_view" name="product.info" template="catalog/product/view.phtml">
14. <block type="catalog/product\_view\_media" name="product.info.media" as="media" template="catalog/product/view/media.phtml"/>
15. <block type="catalog/product\_view\_description" name="product.description" as="description" template="catalog/product/view/description.phtml"/>
16. <block type="catalog/product\_view\_type\_simple" name="product.info.simple" as="product\_type\_data" template="catalog/product/view/type/simple.phtml"/>
17. <block type="catalog/product\_view" name="product.info.addtocart" as="addtocart" template="catalog/product/view/addtocart.phtml"/>
18. </block>
19. </reference>
20. </catalog\_product\_view>
21. </layout>

Disregard the initial XML and layout version declarations. They’re of no significance to us now.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <catalog\_product\_view translate="label">

First, we let the system know that we’re intending to modify the product view part of the system. This is because catalog.xml houses the layout for a number of other views and so it’s imperative that we specify which view we intend to modify.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <reference name="root">
2. <action method="setTemplate"><template>page/1column.phtml</template></action>
3. </reference>

Now, we tell Magento to load up the 1column.phtml file as the main master template for this view. This is because each individual view can use any predefined structure. For example, your home page could use a very complex custom structure, your product page a dual column and your search page a single column layout.

If nothing is specified, it’ll load up the default template mentioned in page.xml. Since we’re using it for everything else, this part is redundant but when you’re modifying this template for your personal use, editing the file’s name is a lot easier than adding chunks of XML to the layout file.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <reference name="head">
2. <action method="addCss"><stylesheet>css/product.css</stylesheet></action>
3. </reference>

And now we come across one of the niftier parts of Magento. Yes, we can throw all our view specific CSS into one giant CSS file but we aren’t cavemen, are we? Cries of multiple HTTP requests aside, this method lets us streamline our CSS better.

First we acquire a reference to the head section of the file and then insert our page specific content into it. Here, I’m inserting a file called product.css which contains all the page specific CSS we looked at above.

Note that you aren’t limited to CSS. Including JS and other assets is possible too.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <block type="catalog/product\_view" name="product.info" template="catalog/product/view.phtml">

We now ask Magento to use a specific template for the content portion of the page

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <block type="catalog/product\_view\_media" name="product.info.media" as="media" template="catalog/product/view/media.phtml"/>
2. <block type="catalog/product\_view\_description" name="product.description" as="description" template="catalog/product/view/description.phtml"/>
3. <block type="catalog/product\_view\_type\_simple" name="product.info.simple" as="product\_type\_data" template="catalog/product/view/type/simple.phtml"/>
4. <block type="catalog/product\_view" name="product.info.addtocart" as="addtocart" template="catalog/product/view/addtocart.phtml"/>

This block defines all the individual blocks inside the main content block. Essentially, we use individual templates for displaying the product’s image, overview and description, availability/price and finally the add to cart functionality.

And with this, our base catalog.xml file is complete.

## Step 4 – Creating our Main Template

Ok, now that we’ve specified our layout we can move on to creating the catalog/product/view.phtml file that we specified as the main template for the content section in the XML earlier.

This file is slightly different from the skeleton template we created in the last part because we’ll be adding some small things directly instead of going through the template path to avoid bloat along with a smidgen of API calls to help us.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php
2. $\_helper = $this->helper('catalog/output');
3. $\_product = $this->getProduct();
4. ?>
6. <form action="<?php echo $this->getAddToCartUrl($\_product) ?>" method="post" id="product\_addtocart\_form"<?php if($\_product->getOptions()): ?> enctype="multipart/form-data"<?php endif; ?>>
7. <div class="no-display">
8. <input type="hidden" name="product" value="<?php echo $\_product->getId() ?>" />
9. <input type="hidden" name="related\_product" id="related-products-field" value="" />
10. <div id="main-product-image"><?php echo $this->getChildHtml('media') ?></div>
12. <div id="product-details">
13. <?php echo $this->getChildHtml('product\_type\_data') ?>
14. <?php echo $this->getChildHtml('addtocart') ?>
15. </div>
17. <h1><?php echo $\_helper->productAttribute($\_product, $\_product->getName(), 'name') ?></h1>
19. <?php if ($\_product->getShortDescription()):?>
20. <h2><?php echo $this->\_\_('Quick Overview') ?></h2>
21. <p class="quick-overview"><?php echo $\_helper->productAttribute($\_product, nl2br($\_product->getShortDescription()), 'short\_description') ?></p>
22. <?php endif;?>
24. <?php echo $this->getChildHtml('description') ?>
25. </form>
26. </div>

If you take a quick look, you’ll notice that we’re making a number of getChildHtml calls to acquire that blocks content. These work just like as expected and the contents of these contents will be covered a little later below.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php
2. $\_helper = $this->helper('catalog/output');
3. $\_product = $this->getProduct();
4. ?>

At the moment, we don’t need to know what this means. In layman’s terms though, we’re essentially asking Magento for this specific product’s store information so we can process it and then display it on the page.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <form action="<?php echo $this->getAddToCartUrl($\_product) ?>" method="post" id="product\_addtocart\_form"<?php if($\_product->getOptions()): ?> enctype="multipart/form-data"<?php endif; ?>>

Here we use one of Magento’s API methods to dynamically generate the URL the form needs to post to.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <h1><?php echo $\_helper->productAttribute($\_product, $\_product->getName(), 'name') ?></h1>

We use another of Magento’s methods to acquire the product’s title directly.

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php if ($\_product->getShortDescription()):?>
2. <h2><?php echo $this->\_\_('Quick Overview') ?></h2>
3. <p class="quick-overview"><?php echo $\_helper->productAttribute($\_product, nl2br($\_product->getShortDescription()), 'short\_description') ?></p>
4. <?php endif;?>

And again, since the quick overview is going to be just a sentence or two, I’m opting to import it directly instead of using a template. We use the same function we used to acquire the title.

## Step 5 – Creating the Templates for our Blocks

With all the main parts done, we can concentrate on the individual blocks now. Now that we’ve already dealt with this when constructing the main template, this should be a lot easier now.

We’ll tackle each in order of appearance in our code:

### Product image

getChildHtml(‘media’) maps directly to catalog/product/view/media.phtml. Our file looks like so:

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php
2. $\_product = $this->getProduct();
3. $\_helper = $this->helper('catalog/output');
4. ?>
6. <?php
7. $\_img = '<img id="image" src="'.$this->helper('catalog/image')->init($\_product, 'image').'" alt="'.$this->htmlEscape($this->getImageLabel()).'" title="'.$this->htmlEscape($this->getImageLabel()).'" />';
8. echo $\_helper->productAttribute($\_product, $\_img, 'image');
9. ?>

Some simple PHP code. We use the helper methods to acquire the product’s image and then render it on screen.

### Availability/Price

getChildHtml(‘product\_type\_data’) maps directly to catalog/product/view/type/simple.phtml. Our file looks like so:

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php $\_product = $this->getProduct() ?>
2. <div id="product-availability">Availability
3. <?php if($\_product->isSaleable()): ?>
4. <span class="available"><?php echo $this->\_\_('In stock') ?></span>
5. <?php else: ?>
6. <span class="unavailable"><?php echo $this->\_\_('Out of stock') ?></span>
7. <?php endif; ?>
8. </div>
9. <div id="product-price">Price <span><?php echo $this->getPriceHtml($\_product) ?></span></div>

We first check whether the item is available or not and then output the required HTML. Acquiring the price of the product is a simple method call away!

### Add to cart

getChildHtml(‘addtocart’) maps directly to catalog/product/view/addtocart.phtml. Our file looks like so:

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php $\_product = $this->getProduct() ?>
3. <?php if($\_product->isSaleable()): ?>
4. <button title="<?php echo $this->\_\_('Add to Cart') ?>" class="button btn-cart"><?php echo $this->\_\_('Add to Cart') ?></button>
5. <?php endif; ?>

As is logical, we check whether an item is up for sale before we output the required HTML. It’s a simple button as is apparent.

### Product Description

getChildHtml(‘description’) maps directly to catalog/product/view/description.phtml. Our file looks like so:

[view plaincopy to clipboardprint?](http://net.tutsplus.com/tutorials/php/magento-for-designers-part-5/)

1. <?php $\_description = $this->getProduct()->getDescription(); ?>
2. <?php if ($\_description): ?>
3. <h2>Product Description</h2>
4. <div class="product-description"><?php echo $this->helper('catalog/output')->productAttribute($this->getProduct(), nl2br($\_description), 'description') ?></div>
5. <?php endif; ?>

Similar to how we included the quick overview, we use Magento’s inbuilt methods to acquire the necessary information. We do check for the description’s existence before we render it.

In case you’re curious, I typically tend to include the description separately via templates mostly because in production templates you’ll almost always be post procession the description in some way. In those scenarios, it’s easier to split this logic off onto its own template. A quick overview, on the other hand, are usually quite small and thus are included directly.

## What We’ll be Building in the Next Part

.. is completely up to you. There are quite a number of views you could design but it’s rather presumptuous of me to pick one. Thus I’m asking you, the reader, to suggest a view to cover in the next part through a comment in the comments section below. Don’t forget to tell me which one to cover next!

## The Last Word

And we are done! Today, we created the first view of our custom Magento theme, the product view along with looking a how Magento lets us add page specific assets and content. Hopefully this has been useful to you and you found it interesting. Since this is a rather new topic for a lot of readers I’ll be closely watching the comments section so chime in there if you’re having any doubts.

Questions? Nice things to say? Criticisms? Hit the comments section and leave me a comment. Happy coding!