Magento Design Guide

Version 1.0

Submitted by

TheUnical Technologies

theunical.com | blog.theunical.com

info@theunical.com | support@theunical.com

All the content of this document is taken from Magento Wiki, Blog and other references. The Purpose of this document is to help those new users.

Table of Contents

Designer's Guide to Magento

Reference:

http://www.magentocommerce.com/design_guide

You have heard about the unbelievable open-source features Magento offers right out the box.You spare no time seeing it in action, have oo-ed and aah-ed over it all - over and over again.

...And now you want to make it all your own. You've got your PSDs ready and the zeal to build and integrate. So what's next—Where do you begin?

Designer's Guide to Magento has been prepared for you to learn and expand your knowledge of the structural, conceptual workings and methods of designing for Magento—It will basically teach you what you need to know to begin creating a theme of your own with Magento. This said, due to the sheer extent of flexibility that Magento's features offer, it is not possible to document all the different ways in which it can be customized for use. For help with this, you can consult Magento's community forums and wiki where real-life people with real-life experiences will be able to point you in the right direction. Also, remember that Magento is an application constantly in development, therefore despite our greatest efforts this documentation may not faithfully reflect the release version you are currently working with.

This documentation is largely sectioned into four chapters and can be skipped through back and forth in order to quickly access only the information you need most. However, because each chapter acts as a prelude to the next, we advise you to follow along with the documentation in the order it was written

Magento Design Terminologies

In order to follow along with the documentation, it is crucial that you have a good grasp of the terminologies used to describe aspects of the Magento system. The terminologies introduced in this chapter are most likely new territory for you, so take your time and read through them thoroughly. But most importantly, don't be discouraged if you can't fully grasp the concept of all these new terminologies - This chapter merely serves to introduce them to you all at once, and further chapters will dig deeper into and expand upon those simple definitions. Terminologies covered in this chapter are:

- Website, Store and Store View
- Interface
- Themes
    - Layouts
    - Templates
    - Skins
- Blocks
    - Structural Blocks
    - Content Blocks

Website and Store

A website is a collection of stores that share the same customer and order information as well as shopping cart. A store is a collection of store views. These are very broad terms that can be adopted to define the unique needs of individual merchants. A few scenarios to define the different uses of website, store and store views are as follows:

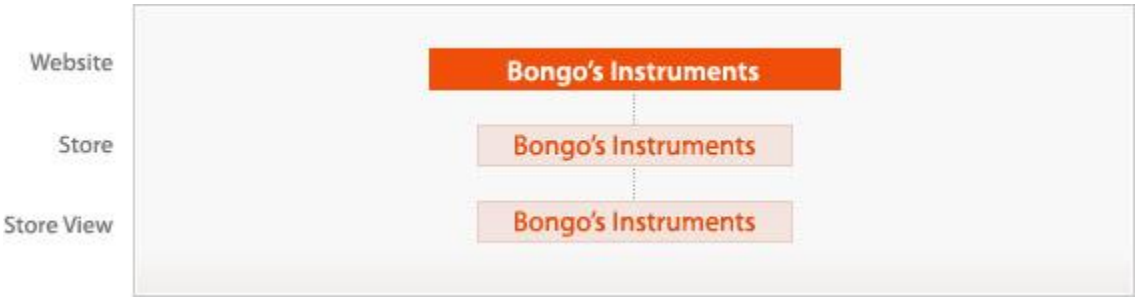
Scenario 1

A company called Dubloo Inc creates an online presence with three separate clothing stores that each cater to different price-level audience. Dubloo Inc wants the ability for all three of its stores to share customer and order information. In such scenario, Dubloo Inc would have one website and three stores under their online presence. *Store* would define the individual price-level store, and *website* would be the Dubloo Inc umbrella.

Scenario 2

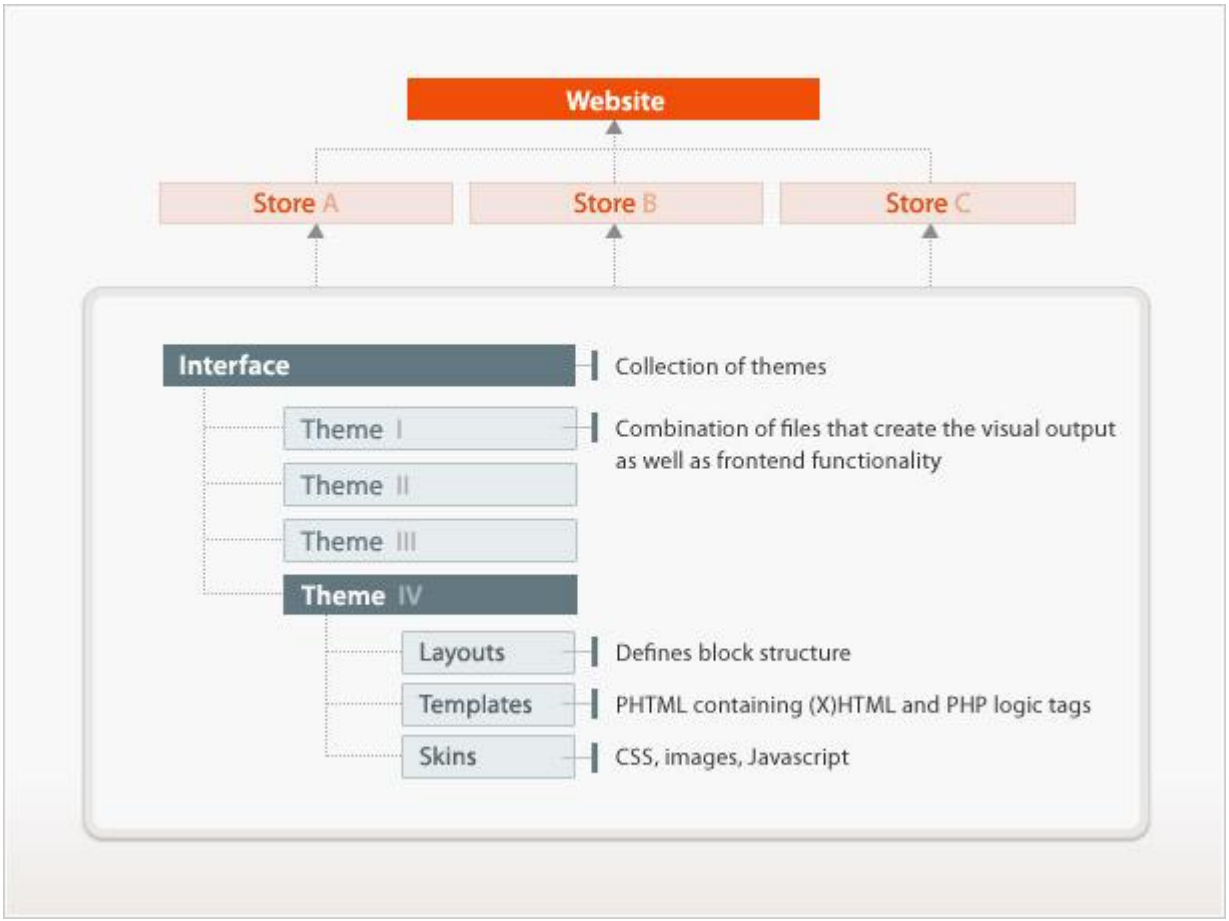
A company called My Laptops wants to open two separate websites that both sell laptops but at differing prices. They also want to offer English and Spanish language options per site, each carrying its own selected items according to language selection. They also need to synchronize customer and order information per site. In such scenario, *store view* would define each English and Spanish branch under the according website. A *website* would define "My Laptops" and "Cheap Laptops".

Scenario 3



A company called Bongo's Instruments wants to create an online presence. With no other branch of stores, Bongo's Instruments is the *store* as well as the *website*.

Interface

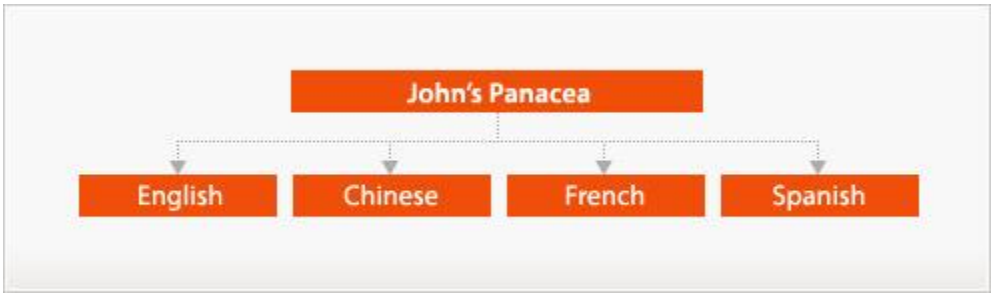


Interface is a collection of themes that determines the visual output and frontend functionalities of your store. An interface can be assigned on either the website-level and/or store view-level through the admin panel (Learn how to assign an interface to the website/store).

If you assign an interface in the website-level, all your stores will inherit the interface of your website. You can further assign an interface in the store view-level and store-, effectively overriding that of the website. Say you

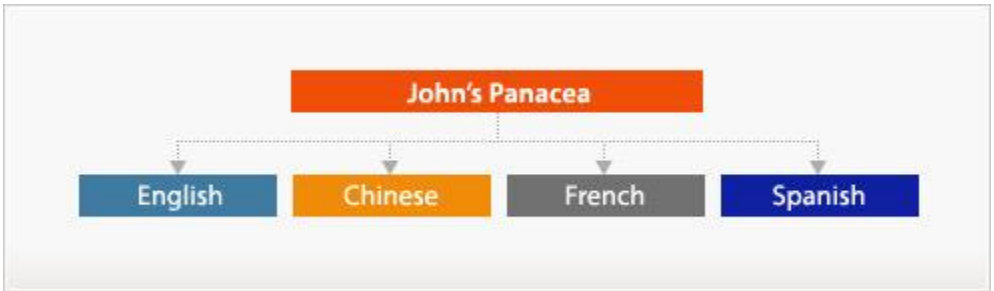
operate four different stores under a website called “John's Panacea” - By studying the effects of each method as shown below, you can easily determine the method to employ for the design needs of your business.

1. *Website-level declaration*



If you want to create a unifying look and feel for all four stores, you will assign an interface in the website level, in which case all four stores will inherit the interface of the website.

2. *Store view-level declaration*



If you want to incorporate a separate look and feel per store, you can assign an interface per store-view, in which case the four stores will each carry its own unique look and feel.

Themes

A theme is any combination of layout, template, locale and/or skin file(s) that create the visual experience of your store. Magento is built with the capacity to load multiple themes at once, therefore distinguishes themes into two large types:

- **Default theme**
  Every interface comes with a theme called 'default' which is the main theme of an interface. When you assign an interface to your store, the application automatically looks for this theme 'default' and loads it to the front-end. In order to customize your store design, you can either modify this theme alone, or create a non-default theme in addition and load it alongside the default. The default theme must contain all the required layouts, templates and skins to run a store error-free and hence is the lowest theme in the theme hierarchy.
- **Non-default theme**
  A non-default theme can contain as many or as little theme files as you see fit for your need. This type of theme is intended for use on creating temporary seasonal design changes to a store without having to

create a whole new set of default theme—By creating a few images and updating some of the CSS, you can easily turn your store from a real bore to a stand-out seasonal Christmas store.

A theme consists of any or all of the following:

- **Layout** (located in app/design/frontend/your_interface/your_theme/**layout**/)
  These are basic XML files that define block structure for different pages as well as control META information and page encoding.For in-depth look into layouts, read Intro to Layouts)
- **Templates** (located in app/design/frontend/your_interface/your_theme/**template**/)
  These are PHTML files that contain (X)HTML markups and any necessary PHP tags to create logic for visual presentation.
- **Locale** (located in app/design/frontend/your_interface/your_theme/**locale**/)
  These are simple text documents organized on a per language basis that contain translations for store copy.
- **Skins** (located in skin/frontend/your_interface/your_theme/)
  These are block-specific Javascript and CSS and image files that compliment your (X)HTML.

Blocks

Diagram 1. Structural Block (Indicated in blue)

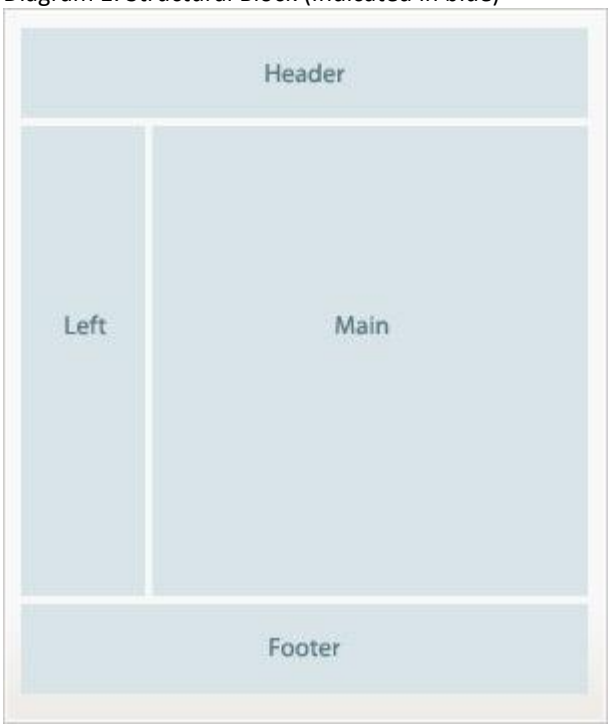
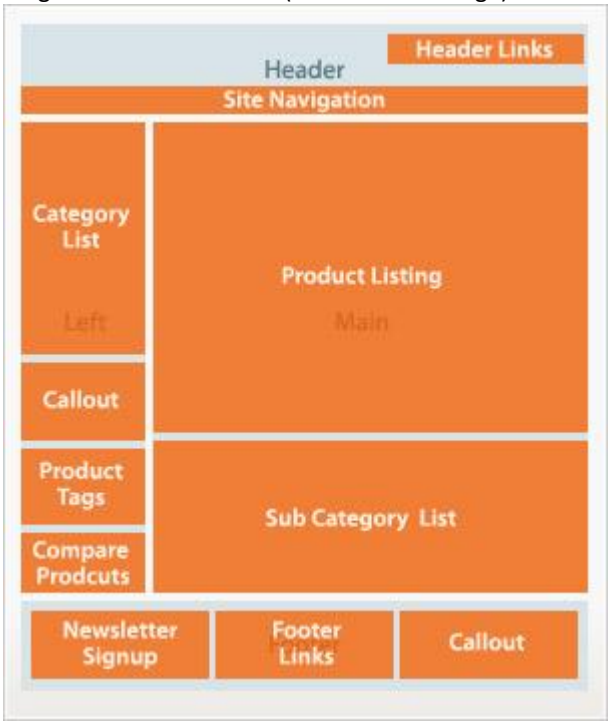


Diagram 2. Content Block (Indicated in orange)



Blocks are a way by which Magento distinguishes the array of functionalities in the system and creates a modular way to manage it from both visual and functional stand point. There are two types of blocks and they work together to create the visual output.

- **Structural Blocks**
  These are blocks created for the sole purpose of assigning visual structure to a store page such as header, left column, main column and footer (Diagram 1).
- **Content Blocks**
  These are blocks that produce the actual content inside each structural block. They are representations of each feature functionality in a page and employs template files to generate (X)HTML to be inserted into its parent structural block. Category list, mini cart, product tags and product listing…etc are each its own content block (Diagram 2).

Instead of including template after template as a typical eCommerce application would in order to gather the whole (X)HTML output, Magento gathers and arranges page content through blocks.

Working with Magento Themes

In this chapter we'll go into the details of how Magento handles themes and teach you how to create and manage your own:

- How Magento does themes different from others
- How to create a theme
- How to assign interface and theme to the store

- Say hello to multiple themes
- Hierarchy of themes: How Magento handles multiple themes

How Magento Does Themes Different from Others

The term 'theme' probably sounds to you a familiar ring. Either as an audience, a creator, or both, you've experienced the sea of available web applications options via its themes. A theme has largely two types of users – First, the audience–type who experiences it from the aesthetical and usability standpoint by browsing through a store – And second, the creator–type who goes through an additional layer of the theme experience by taking part in building the theme.

To the first user type, a store experience is defined by the ability of a store to fulfill his/her tactical and emotional demands. To the second user type who must fill the creator role, it is the efficiency with which he/she is able to complete a set of development tasks that determines the store experience.We recognize that it is the accumulated experiences of both user types that determine the final profitability of the store, therefore neither user's experience can afford to be overlooked.

Because we know that as designers you've already got the graphical end of things best covered (servicing to the audience–types), we figured we'd just help you along by building **an out-of-this-world theme management to maximize your work-flow efficiency and take your creativity to the next level.** Here's some things we thought might get you drooling:

1. **Maximum customization power**
   With Magento you can update the look and feel of your store in the category and product level, giving you greater marketing and promotional power as well as a store with endlessly unique design. Ever imagined presenting each product in its own customized product info page? Magento gives you the power to do just that and more, by providing a quick way to customize your product presentation in a per-product and per-category basis.
2. **Multiple themes**
   Magento gives you the ability to load multiple themes at once, allowing you to swap between a default store design and temporary event/season-specific ones — All at the command of a few key strokes.
3. **Uninterrupted workflow**
   With Magento's fully object-oriented programming, all modules are immediately accessible via template tags from any template files. And because Magento comes feature-rich right out of the box, you'll never again have to be dependent on a programmer to finish the simplest tasks for you. Magento also thrives on an extensive network of knowledgeable community members (including the official Magento Team), so you will never need to think twice about where to get guidance should you need it along the way.
4. **Minimize debugging time**
   Any designer can recall those precious hours and minutes wasted looking for the unclosed markup scrutinized by your validator. The validator may tell you what's wrong, but it never seems to tell you where it's happening. Magento's modular backend brings with it a modular template system that minimizes the amount of (X)HTML you need to handle at once. Less the mess means less the fuss and more the sanity for the truly important things in life.

Really, the best part of what Magento offers you though, is an application that thrives on its flexibility, leaving you with nothing much to worry about except devising your brilliant plans for your next 'wow' front-end. The sky is the limit for this application, and we hope you have fun applying it to your store.

How to Create a Theme

Let's first unveil some directories to get you going. Open the following directories in your Magento root and do get nosy:

- **Directory 1:** app/design/frontend/default/default/ — This directory contains the layout, translation (locale) and template materials.
- **Directory 2:** skin/frontend/default/default/ — This directory contains the images, CSS and block-specific Javascripts.

When working with themes, these two directories will remain your base starting point.
As you may have noticed, we've sectioned the theme files into two parts. By separating the files that must be web accessible (such as image and Javascripts) from those that can be hidden from it, we've made certain Magento offers you maximum security measures for your store at every corner.

Let's go ahead and examine the two directories.
At first blush you'll notice the use of directory names "default/default" in both directory 1 and 2 like so:

- **Directory 1:** app/design/frontend/**default**/**default**/
- **Directory 2:** skin/frontend/**default**/**default**/

In both cases, ***** indicates the *interface* name, and ***** indicates the *theme* name. So if you were working on a theme called "my_theme" in an interface called "my_interface", you would be working in the 'app/design/frontend/my_interface/my_theme/' directory.

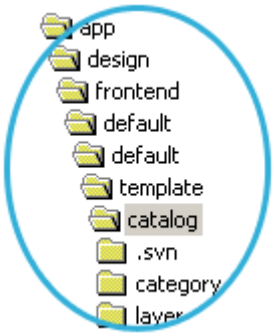
You can save as many themes into your interface directory as you'd like, but at the time of writing, your store can only handle loading the theme called 'default' and one additional theme of your choice to your store. (**Note:**Capacity to load unlimited number of themes will become available in the coming releases.) To learn how Magento manages loading multiple themes at once, read Hierachy of Themes) This gives you two themes to work with, so you can keep your default theme for your store's off-season design and use the additional theme for the seasonal and event-specific ones (To read about how multiple themes work and how you can benefit, read Say Hello to Multiple Themes).

Creating a new default theme

In order to create a new default theme, first you must copy an existing default theme from which you can base your new theme. Create a duplicate of app/design/frontend/default/default/ and rename the new directory to whatever you would like (Name of an interface and theme should be one alpha-numeric word starting with a letter from the alphabet[a-z]. For instance, 'My New Theme' and '02123_my_theme' is bad. 'my_new_theme' is good). The name of your theme directory is the name by which the application will recognize your theme. Now do the same for skin/frontend/default/default. And that's it! You've now successfully created a new default theme. To learn how to assign this theme to your store, read Assigning a new theme below.

Creating a new non-default theme

Diagram 1

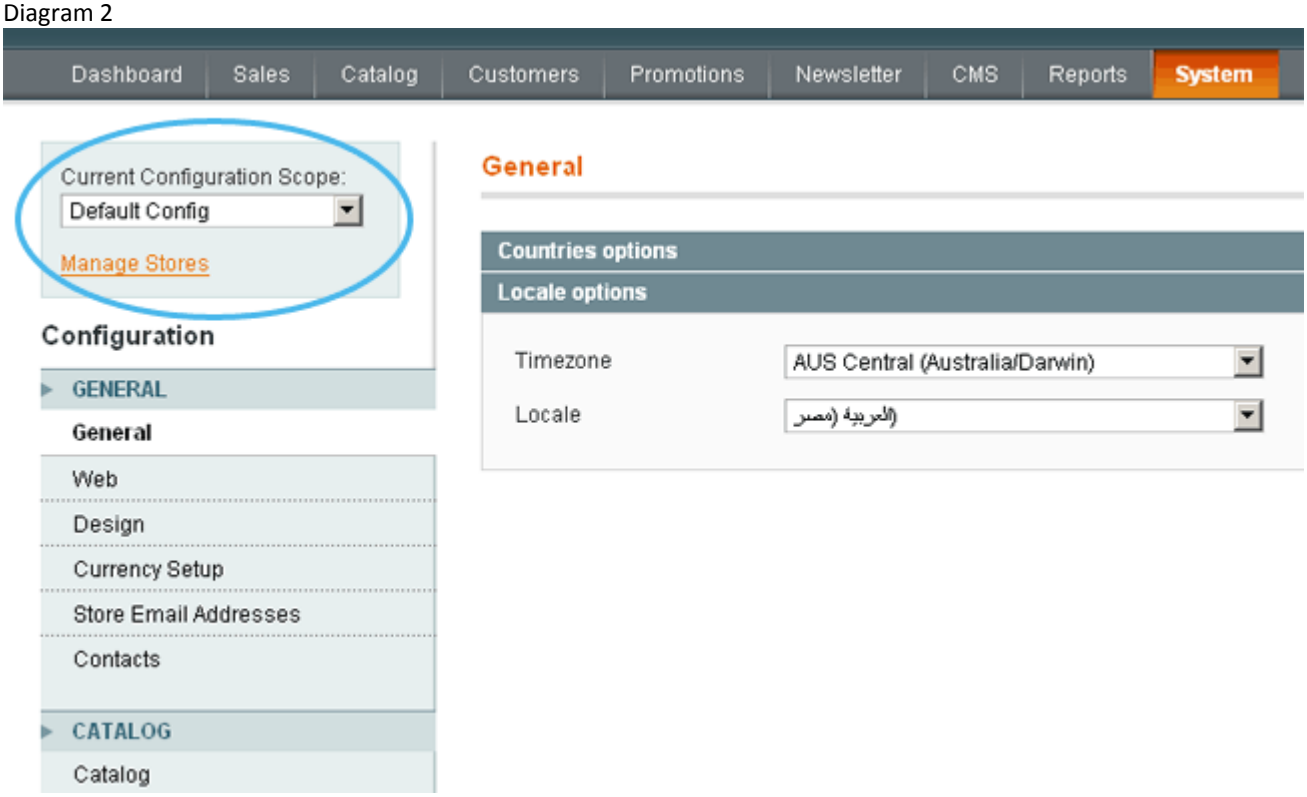


When creating a new non-default theme, you don't need to duplicate any existing default theme directories. Most likely you're just making changes to specific files and hence will only need to duplicate the according files as a starting point of your new theme. One rule you must always remember to follow, however, is to make certain that you preserve the subdirectory structural conventions of Magento. For example, if all that your non-default theme contains is the template file 'home.phtml' from the catalog module, inside app/design/frontend/your_interface/your_non_default_theme/ you will need to create directories 'template/catalog' into which you will save your template file. When you open up a default Magento theme directory (Diagram 1), you will see how directories are structured—Make sure to reference this directory convention in order for your new theme to load successfully.

Assigning interface and theme to the store

Now that you've created your own theme (whether a default or a non-default), you'll need to assign it to your website/store in order for it to take effect. Navigate to the Magento admin panel (ie.www.mydomain.com/**admin**), then the Design configuration tab (System -> Configuration -> Design tab).

Diagram 2



On the upper corner of the left column in Diagram 2, you will see a box labeled 'Current Configuration Scope'.

- In order to **manage your store design in the website-level**, select the name of your website from the dropdown, then apply the following steps.
- In order to **manage the design from the store view-level**, select the name of your store view from the dropdown, then apply the following steps.

Step 1
From the Design tab, in *Current package name*, enter the name of the interface in which your new theme resides. Magento will automatically load the interface called 'default' if this box is left blank.

Step 2
In *Default* (under Themes heading), enter the name of the new theme you'd like to load to your website/store. If you leave this box blank, Magento will automatically load only the theme called 'default' (Remember, no matter how you configure the design tab, Magento will automatically load the theme called 'default'. If you assign a theme in the admin, that theme will simply load higher up in hierarchy, but will load along with the theme 'default'. This will change in the upcoming stable release however, to give you full control over what themes are loaded into the store). Should you choose to load the theme separately depending on file type (layout, templates, skin or translation files), enter the name of the theme from which to grab the according file types.

Step 3
When you're done, click the button "*Save config*" and reload your store – Voila! You now see your new theme reflected in the frontend.

Now that we've got the how-to of creating and managing themes, let's move on to how Magento handles those themes.

Say Hello to Multiple Themes

Note: Capacity for unlimited themes will become available with the upcoming releases. Although at the time of writing only two themes can be loaded at once, the workings behind the scenes remain the same and you will benefit from this documentation.

Holiday seasons by far offer the most extensive sales opportunity for any eCommerce store — Customers line up to buy Christmas gifts for their family and friends, and moms line up to buy Halloween costumes for their child's special night of trick 'or treating. In order to tailor to the seasonal shoppers, your store must faithfully reflect the occasions in order to inspire your shoppers to explore your store. A shop like Diagram 3 just doesn't cut it during Christmas – What this store needs is a few reds, snowflakes and Santa Claus – just like the store below in Diagram 4!

Diagram 3. This just doesn't cut it

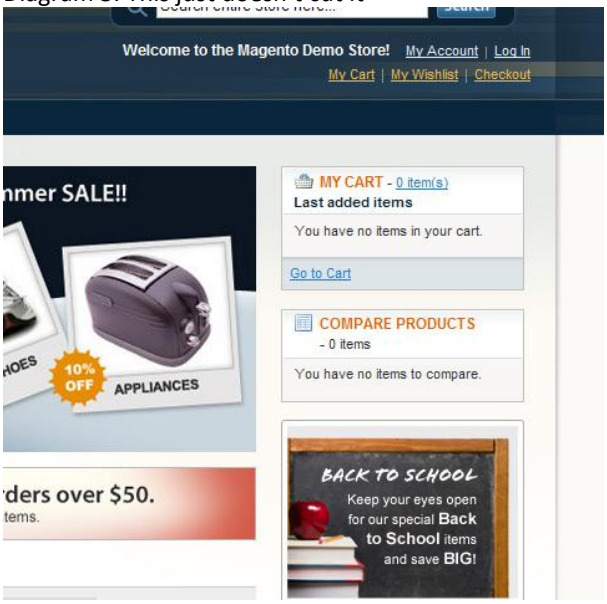
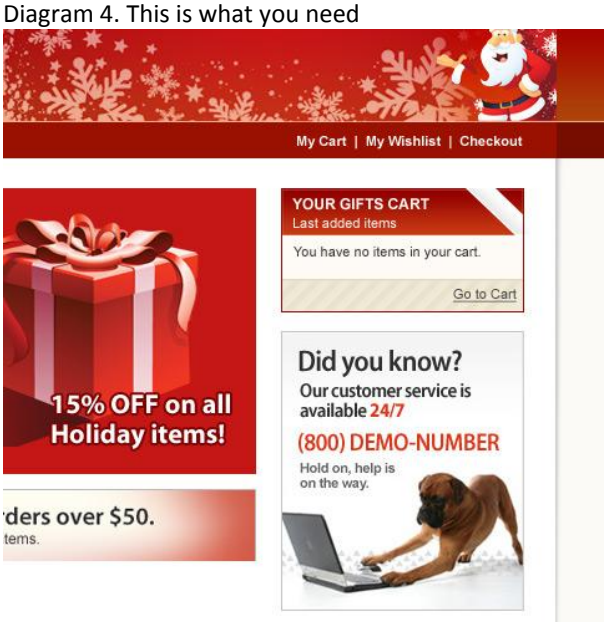


Diagram 4. This is what you need



With Magento, we've created the capacity for your store to handle multiple themes of your choice exactly for those times when an extra touch is needed most. By loading multiple themes to your store, you can preserve your normal non-seasonal store design, while enhancing it with the Christmas theme.

At close examination of the two designs above, you'll notice similarities in the underlying design. The façade has gone Christmas in Diagram 4, but underneath it you can still see the structure of the off-season store design. The only real difference between the two store designs, are just a few CSS and image files and wording changes in the template files. Because the changes are in fact minor, you don't need a whole new default theme to accommodate your Christmas theme. What you need is just a few file replacements, and you're on your way to a much spicier and merrier store. Magento's multiple themes functionality was created to accommodate exactly that need, putting the power on your fingertips to turn on and off the seasonal themes while preserving your default theme.

Magento handles loading multiple themes by assigning something called *hierarchy* to the themes which is simply a process of cancelling out redundant files to load only that which resides highest in hierarchy.The hierarchy is determined by you in the Admin Design configuration tab and your ability to do so will become fully functional with the upcoming stable version. At the time of writing, the hierarchy of themes is already decided for you, as Magento loads the 'default' theme in the system first (placing it in the lowest hierarchy), then loading the second theme you assign in the admin (placing it in highest hierarchy).

Hierarchy of Themes

When you assign multiple themes to your store, you're essentially taking advantage of the fact that while *your* main goal when building a theme is to create the most usable and visually pleasing graphical interface, the goal of *Magento's* is to ensure that the application is able to locate and load all the files of the theme required to keep the application running error-free.

For instance, if your category listing page calls for a template called 'view.phtml'(in which case this template becomes a required file), but the application is unable to find the file in the theme highest in hierarchy (**Note:** At the time of writing, the theme highest in hierarchy is the theme that you assign through the admin, and the theme

lowest in hierarchy is the theme called 'default' that Magento automatically loads into the store. In the upcoming releases, Magento will provide you with the ability to have full control over the hierarchy of your themes), it will look to the next theme highest in hierarchy to find the file. Should this fail, it will continue working down the hierarchy of themes until it's able to locate the file called 'view.phtml', upon which the application will load it to the store and terminate the search. This method of building design is called *fallbacks*, because the application 'falls back' to the next possible source of required files in order to retrieve and load it.

Let's say you have three themes assigned to your store and each of these themes contain the following files:

Table 1

default	**my_theme_1**	**my_theme_2**
All required files	templates/3-col-layout.phtml	templates/3-col-ayout.phtml
	templates/header.phtml	css/base.css
	images/logo.gif	
	css/base.css	
	css/boxes.css	

Let's also assume that the three themes are assigned this hierarchy:

Table 2

HIGHEST	my_theme_2
	my_theme_1
LOWEST	default

At close observation, you'll see that there're few redundant files such as templates/3-col-layout.phtml and css/base.css in Table 1. Now let's arrange the table around so the redundant files are arranged parallel to one another.

Table 3

default	**my_theme_1**	**my_theme_2**
All required files		
	templates/3-col-layout.phtml	templates/3-col-layout.phtml
	templates/header.phtml	

images/logo.gif

css/base.css css/base.css

css/boxes.css

'Ok, great. But what does this mean?' you may ask.
Well, let us remind you that the files in Table 3 are how *you* see the files in each theme folders and not how *Magento* sees it.

Let's then look at how *Magento* sees the same file structure in Table 4.

Table 4

default	**my_theme_1**	**my_theme_2**
All required files		templates/3-col-layout.phtml
templates/header.phtml		
images/logo.gif		
		css/base.css
css/boxes.css		

You'll notice how Magento ignores the version of the redundant file lower in hierarchy and recognizes only the version higher in hierarchy. This is because it's already found the required file and need not search for it any longer, ergo terminating the search for that specific file while continuing the search for other required files yet to be found.

Building Your Theme

Magento is built on a fully modular model that transfers to unlimited scalability and flexibility for your store. By nature of the application, this approach to programming is mirrored in the way you will develop themes for your store. In this chapter, we will explore what that means for you and exactly how to go about developing a theme for your store with Magento.

Introducing Blocks and Layout

You've most likely worked with other eCommerce applications before arriving at Magento—with such experience we know you probably have a few sets of things that have become second nature to you when developing a theme for your store. Before you begin this documentation, we'd like to make sure you gather those expectations you may have and mercilessly throw them out the window. No, no, this doesn't mean you need to learn a whole new

language. It doesn't even mean much will change with regards to your workflow, actually. All this really means is that there're a couple of new tricks you will need to learn and along with it new tools we'd now like to introduce to you. Marry these tools, keep them at your side at all times and make sure you pay them lots and lots of attention—You do all this, and we promise you, you will love yourself for it. Are you ready? Here they are:

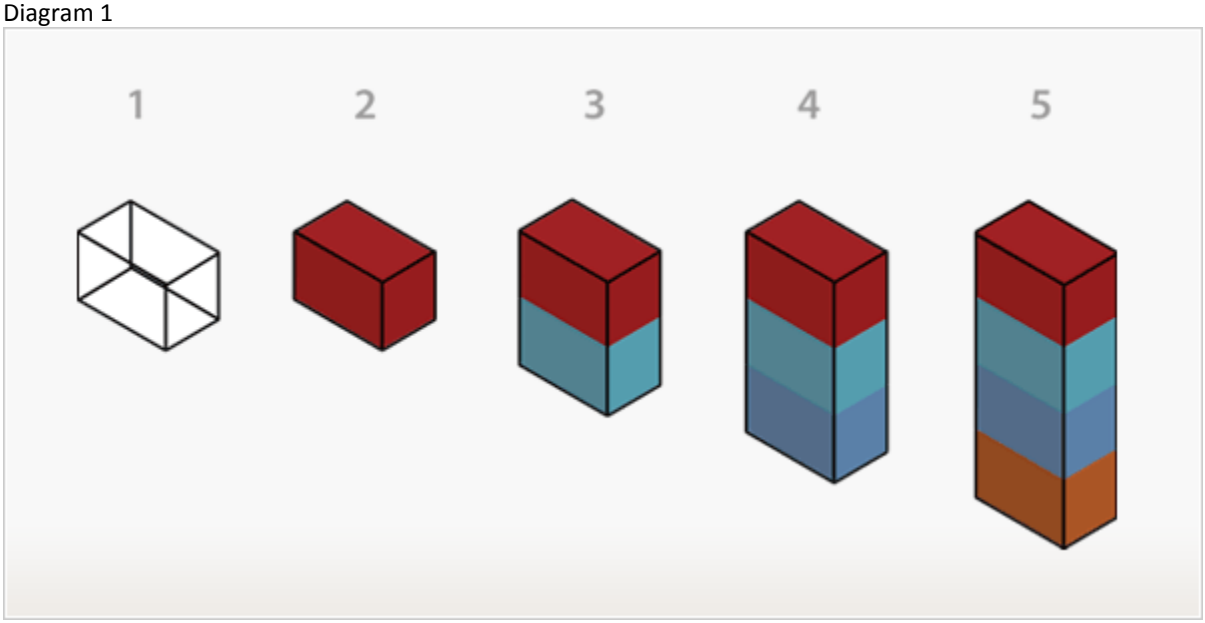
1. **Structural Blocks**
2. **Content Blocks**
3. **Layout**

Creating the Mental Note

In order to give you a clear understanding of what blocks and layouts are, here's a mental picture you can draw for yourself:

1. Imagine an outline of a block (like in Diagram 1 -- Don't reference the picture the whole time. Imagine it and follow along).
2. Now imagine the outline of a block filled with one block.
3. Now imagine two blocks and the outline growing with the blocks.
4. Now imagine three blocks and the outline growing with the blocks.
5. Now imagine four and the outline growing with the blocks.

Diagram 1



6. Now look at this snapshot of the category landing page in Magento:

Diagram 2

7. And now look at the break-down of the above snapshot into two types.

Diagram 3

Diagram 4

The mental picture you just created serves to give you a parallel comparison between concept and actual realization of blocks. I know you're probably completely lost. Let us explain.

In concept, the outlines in Diagram 3 are the **structural blocks**. They are the parent blocks of content blocks and in realization, serve to *position* its content blocks within a store page context (as in Diagram 4). These structural blocks exist in the forms of the header area, left column area, right column…etc. which serve to create the *visual structure* for a store page.

A **content block** on the other hand, in concept, is the individual-colored blocks that make up a structural block. In a store context, they are the true meat of a store page. They are representations of each functionalities featured in a page (such as category list, callout and product tags…etc.), and employs template files to generate (X)HTML to be inserted into its parent structural block.

Layout is the tool with which you can assign content blocks to each structural block you create. Layout exists in the form of XML text-file and by modifying the layout you are able to move blocks around in a page and assign templates to the content blocks to produce markup for the structural blocks. In fact, with the help of a few layout files alone, you are able to modify the visual layout of every page in your store. Read further about layouts in Intro to Layouts.

With Magento, you will no longer have a template file called left_column.ext and in it have the never-ending spaghetti of markups that must be manually managed depending on each functionality needed for the page. Instead, your templates are managed on a per-functionality basis and you can load and unload functionalities in your store page by the virtue of a couple of layout commands alone.

Step by Step Guide to Building a Theme

Here're the full list of tools Magento gives you to build your themes:

1. **Templates**
2. **Layouts**
3. **Blocks**
4. **Skins** (images, CSS and block-specific Javascript)

In order to build a theme for your store, your work flow will consist of the following steps:

ONE: Disable your system cache

In order to prepare your Magento for production, you need to first disable system cache by going to the Administration Panel (http://yourhost.com/**admin**) and navigating to System -> Cache Management. Select 'Disable' from the 'All Cache' select box and click on save. Doing this will ensure that you see a faithful reflection of your store front as you make the changes.

TWO: Determine all the possibilities of structure types for your store

Before you even start creating the markup for the store, you will first need to ask yourself the type of page structure you'd like to have for each of your store pages. Make yourself a list that looks something like this:

- Home page will use the *three column structure*.
- Category listing page will use the *two column structure that includes a right column*.
- Customer pages will use the *two column structure that includes a left column*.

Skeleton template

Once your list is complete, you will create the (X)HTML markups for each structure type and save them as *skeleton templates* into app/design/frontend/your_interface/your_theme/template/page/. A skeleton template is called such due to the nature of its purpose—all it really contains (aside from the <head> elements), is the presentational markups that serve to position each *structural block* into according markup areas.

Sample skeleton template

```
<html>
<head></head>
<body>
<div class="header"><?=$this->getChildHtml('header')?></div>
<div class="middle">
    <div class="col-left"><?=$this->getChildHtml('left')?></div>
    <div class="col-main"><?=$this->getChildHtml('content')?></div>
</div>
<div class="footer"><?=$this->getChildHtml('footer')?></div>
</body>
</html>
```

Upon scanning through the sample skeleton template above, you will notice a PHP method called <?=$this->getChildHtml()?> inside each presentational markup. This is the way Magento loads structural blocks into skeleton templates and hence is able to position all the contents of the structural blocks within a store page.

Each getChildHtml calls on a name as in <?=$this->getChildHtml('**header**')?> , and these names are ways by which each structural block is identified in the layout. Skeleton templates are assigned to the store through the layout.

THREE: Cut up your (X)HTML according to functionality

Once you've created your skeleton templates, you will now need to create the template for each *content block*.

Magento likes meaningful templates

You will need to cut up the (X)HTML markup built for your page and supply the according markup to each functionality of the page. For instance, if you have a mini-cart area in your design, the markup for this area alone will become its own template file. Same for your product tags, newsletter sign up…etc. All these functionalities already come with Magento, so you can reference the existing template tags to build your mark-up logic.

Where to save the templates

Diagram 5

The full markup for any page in your store is introduced via an array of templates that each represents a functionality of a module. In order to find out what templates are being called to a page you'd like to modify, you can turn on the Template Path hints. In order to enable it:
1. Go to the admin and navigate to System -> Configuration
2. Select your store in the top left website/store selector
3. When the page reloads, select the 'Developer' tab and select 'Yes' for Template Path Hints.
When you're done, go back to the store front, reload your page and you will see the path to all the templates listed according to the block. In order to modify the markup, all you have to do is follow the path written out for you and modify the according template(s).

FOUR: Change the layout to reflect your design

Once you've distributed some of the markups, you probably would like to now move the templates around in a page to see how you're progressing along.

Where to find layouts

In order to access the layout files, go to app/design/frontend/your_interface/your_theme/layout/. Just like the templates, layouts are saved on a per-module basis therefore you can easily locate the layout file to modify with the help of the Template Hints. First, enable Template Hints, reload the the page you want to modify, and look at the path of the template file(s) that the Template Hints will provide you with. If you want to (for instance) move the mini cart, reference the template path (ex:app/design/frontend/default/default/**checkout**/cart/sidebar.phtml) and use the first folder name inside the theme folder (indicated in bold, which is the module name) to find the according layout file. So in the case of the mini cart, you know to look in '**checkout**.xml' to modify the mini cart positioning. Each layout file(should it be necessary) further sections into per-page basis layout command. Each area of per-page layout is clearly marked with comments reflecting its usage, but the application itself recognizes the layout separation by the handles of each layout.

Default layout versus Layout Updates

There are two types of layouts--default and updates. A *default layout* (page.xml) is the layout that by default applies itself to almost every page in the store. All other layout files are *Layout Updates* that simply *updates* the default layout on a per-page basis.

Let's take for example your skeleton template:
In the default layout, you have it set to three columns, which means by default most all of the page in your store will have the three column page structure. But it's not the three column structure you need for your product page. For *it*, you want a two-column structure that includes a right column. To accommodate this, you will leave the default layout alone and open catalog.xml in which you can place some layout commands that tells the application to load the two-column structure to your product page instead of the default three. This is called the process of *updating* a layout.

Example way of assigning skeleton template

```
<reference name="root">

    <action method="setTemplate"><template>page/2columns-right.phtml</template></action>

</reference>
```

Let's take another example:
Say by default you want a newsletter sign-up box in your right column, but in customer account pages you want to exclude it. In this case, you would leave your default layout alone and open up customer.xml, into which you will place a command that *unsets* the newsletter content block, excluding the newsletter functionality from the page.

Intro to Layouts

Layout, by appearance of its components, can easily deceive you into believing that in order to work with it you must first be armed with an extensive knowledge of programming logics. Nothing can be further from the truth. Layout is built with a small set of XML tags that are easy and fun to learn. By learning some key concepts and commands of layout, you will soon be armed with the confidence and knowledge to easily modify your store design to your desired spec.

How Layout Works

Layout is a virtual component of the Magento application. By modifying the components of layout, you can build your store page in an upgrade-compatible way.

Diagram 1

Layout is comprised of *default layout* and *layout updates* that are made up of easy-to-learn XML tags. With these layout commands, you can modify/assign content block-structural block relationships and also control store-front functionalities such as loading and unloading of block-specific Javascripts to a page.

Layout files are separated on a per-module basis, every module bringing with it its own layout file (for instance 'catalog.xml' is a layout file for the catalog module, 'customer.xml' is for the customer module...etc). These layout files are located in app/design/frontend/your_interface/your_theme/layout/ and each file is further separated by handles (see diagram 1), each handle (with the exception of <default>) assigning its nested updates to the according specific page in the store.

Some layout files may contain the <default> handle. When parsing the layout files, Magento first grabs the layout updates assigned in the <default> handle of almost all layout files, reading them in the order as assigned in app/etc/modules/Mage_All.xml. It then parses the page-specific layout update, finalizing the building of a store page.

The system is built this way in order to allow seamless addition and removal of modules without effecting other modules in the system. Anatomy of Layout

Layout contain a small set of XML tags that act as detailed instructions to the application on how to build a page, what to build it with and the behavior of each building block. The best way to approach layout is to just dive right in and attack it from all angles. In order for you to do that, here're some behavioral properties of each layout XML tag.

Handle

Handle (diagram 1) is an identifier by which the application determines what to do with the updates nested by it.

If the name of the handle is <default>, then the application knows that its nested updates must be loaded on almost all the pages of the store prior to loading page-specific layout (We say 'almost all', because some exceptional pages like the product image popup does not load the layout in the <default> handle).

If Magento finds handles other than <default>, it will assign the updates nested inside the handle to the according page specified by the handle. For instance, <catalog_product_view> contain the layout updates for the Product View page, while <catalog_product_compare_index> contain those for the Compare Product page. Handles are

set-in-stone identifiers that as a designer with no extensive understanding of Magento programming, should never need to modify.

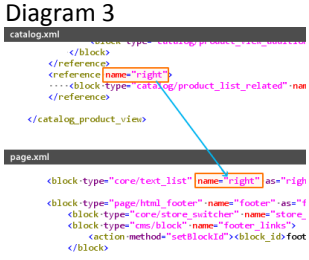
<block>

Magento determines the behavior and visual representation of each building block of a page via the <block> tag. We've already mentioned the two types of blocks Magento employs - *structural blocks* and *content blocks*. The best way to distinguish between the two is by examining the behavior assigned to it via the tag attributes. A structural block usually contains the attribute 'as', through which the application is able to communicate with the designated area (by the getChildHtml method) in a template. You will notice many occurances of this 'as' attribute in the default layout, because by nature the default layout is one that builds the ground work upon which the page-specific layouts can begin adding onto. For instance, in the default layout, there're structural blocks such as 'left', 'right', 'content' and 'footer' being introduced. Not to say these blocks cannot exist in normal layout updates, but why not first set up the reoccurring structural blocks in the default layout first, then start adding content on a per-page basis? Let's dig further into available attributes for <block>.

- **type** – This is the identifier of the module class that defines the functionality of the block. This attribute must not be modified.
- **name** – This is the name by which other blocks can make reference to the block in which this attribute is assigned (see diagram 3).
- **before** (and) **after** – These are two ways to position a content block within a structural block. before="-" and after="-" are commands used to position the block accordingly at the very top or very bottom of a structural block.
- **template** - This attribute determines the template that will represent the functionality of the block in which this attribute is assigned. For instance, if this attributes is assigned 'catalog/category/view.phtml', the application will load the 'app/design/frontend/template/catalog/category/view.phtml template file.In order to learn about how layout works with templates to bring markup to your store, read Step by Step Guide to Building a Theme
- **action** – <action> is used to control store-front functionalities such as loading or unloading of a Javascript. A full list of action methods will soon become available, but in the mean time the best way to learn about the different action methods is by playing around with them in the currently available layout updates.
- **as** – This is the name by which a template calls the block in which this attribute is assigned. When you see the getChildHtml('block_name') PHP method called from a template, you can be sure it is referring to the block whose attribute 'as' is assigned the name 'block_name'. (ie. The method <?=$this->getChildHtml('header')?> in the a skeleton template correlates to <block as="header">)

<reference>

Diagram 3

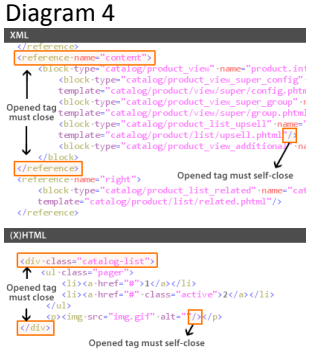


<reference> is used to make reference to another block. By making a reference to another block, the updates inside <reference> will apply to the <block> it correlates to (see diagram 3).

In order to make the reference, you must target the reference to a block by using the 'name' attribute. This attribute targets the <block> tag's 'name' attribute. So if you were to make a reference by <reference name="right">, you're targeting the block called <block name=" right ">.

Rules of XML

Diagram 4



If you're not familiar with it, upon encountering Magento's XML-based layout updates, you may have a question or two about the rules you must follow when modifying an XML file.
The only set rule you need to remember with regards to XML, is that when a tag opens, it must either be followed by a closing tag(<xml_tag></xml_tag>) or self-close(<xml_tag/>) exactly as (X)HTML file tags would.

Quick Exercises to Get You Started

Just like any new concepts, we know that without actually diving in and getting your hand dirty, you can never truly grasp what you've been explained in words. So here are a couple of exercises for you to get a feel for working with layouts. In order to work with this exercise, you must have the default Magento theme ready and accessible.

Exercise #1: In the category page, move the "My Cart" box from the right column to the left

1. Turn on the Template Path Hint by going to the admin then navigating to System -> Configuration. When you're in the configuration page, select the store you're working in by using the top left website/store selector. Wait for the page to reload, then select the Developer tab. Select 'Yes' in the select box for Template Path Hints. Click on Save. Go back to the store front, and reload.
2. When the page reloads, look at the template path of the 'My Cart' block- it'll most likely say 'frontend/default/default/template/checkout/cart/sidebar.phtml'. By looking at the path, you know this template is being introduced via the 'checkout' module. How do you know this? - frontend/default/default/template/**checkout**/cart/sidebar.phtml. It says so in the template path. The immediate directory name following 'template' is the name of the module through which a template is introduced.
3. Open up layout/checkout.xml - because now we know we're dealing with the checkout module
4. Search for 'checkout/cart/sidebar.phtml' (the template name of the My Cart block) in the layout updates. You will find an area that looks like this:
   <reference name="right">
   <block type="checkout/cart_sidebar" name="cart_sidebar" before="-"
   template="checkout/cart/sidebar.phtml"/">
   </reference">
   Change it to say the following instead (Note that all you're doing is changing the <reference name="right"> to <reference name="left">).

<reference name="left">
<block type="checkout/cart_sidebar" name="cart_sidebar" before="-"
template="checkout/cart/sidebar.phtml"/">
</reference">

5. Reload the store front and you will now see the change reflected.

Exercise #2: Separate the SEO links at the footer area - Instead of having the link items to be one list, isolate 'Advanced Search' to be in the header instead.

1. You can make a calculated assumption that the SEO links must be assigned somewhere in the layout under the footer block because it's being pulled through '<?=$this->getChildHtml()?>' of template/page/html/footer.phtml. (You will need the Template Path Hints turned on to see why this is obvious).
2. Open up layout/page.xml and look through the list of children under the footer block in order to locate a block that calls the footer links - You will find <block name="footer_links">, which is what calls the SEO links. Now that you know that layout updates reference the SEO links via the name="footer_links", now you will do a search in all the xml files for <reference name="footer_links">. You will find references for the footer_links block in catalog.xml (which calls 'Site Map'), catalogsearch.xml(which calls 'Search Terms' and 'Advanced Search') and contacts.xml (which calls 'Contact Us').
3. Now that you've located the area of the individual SEO link items, you will now begin the steps to isolate 'Advanced Search' from the bunch and make it it's own thing in the header. First go back to page.xml and create a new block <block type="page/template_links" name="**header_links**" as="**header_links**" template="page/template/links.phtml"/> and nest it inside <block name="header">. You've made the layout updates to expect this link in header.phtml. Open template/page/html/header.phtml, and type in <?=$this->getChildHtml('header_links')?> where you want the link to reside.
4. Now I go to catalogsearch.xml, and cut this:
   <action method="addLink" translate="label title" module="catalogsearch"><label>Advanced Search</label><url helper="catalogsearch/getAdvancedSearchUrl" /><title>Advanced Search</title></action> out from <reference name="footer_links">.
   I create new lines to reference the new header_links block I created, and nest the cut out code inside it like so:
   <reference name="header_links">
   <action method="addLink" translate="label title" module="catalogsearch"><label>Advanced Search</label><url helper="catalogsearch/getAdvancedSearchUrl" /><title>Advanced Search</title></action>
   </reference>
5. Now I have Advanced search in the header instead of the footer.

This marks the end of Designer's Guide to Magento. Hopefully by now you're armed with confidence and knowledge about how to approach designing with Magento. We hope to see some of your results up on the forum. Share with us your designs, discuss this documentation and ask lots of questions at the community forum's 'HTML, XHTML, CSS, Design Questions' thread.

More information links

http://www.magentocommerce.com/media/screencasts/designers-guide-1/view

CSS Resources

Changing your CSS files on the fly

Mozilla does a great CSS Developer plug-in that allows you to change the CSS files on the Fly

Plugin

https://addons.mozilla.org/en-US/firefox/addon/60 great CSS Developer plug

Mozilla

http://www.mozilla.com/en-US/ Mozilla Download if you dont have it

A great CSS book here called THE CSS ANTHOLOGY 101 Essential Tips Tricks and Hacks

http://www.euphorish.com/2007/07/02/html-css-article-overview-lay-out/

How to use Web Developer CSS

open your Mozilla Browser (forget winddoze browsers) locate

TOOLS » Web Developer » css » edit css

Now at the bottom of the browser you should see a **CSS panel**

This will list all the CSS files called into that particular page of you store

select which files contents you need to change IE **boxes.css menu.css etc**

cut all the css out of boxes.css and watch what happens the page in the browser will render with no Style

refresh the browser and it all comes back

The following is a list of CSS resources for those wanting to polish their skills:

http://www.westciv.com/style_master/academy/css_tutorial/ (the number one guide to CSS)

http://www.w3.org/Style/CSS (and the boring docs, but nevertheless... some good detailed info here)

http://www.nypl.org/styleguide/ (XHTML and CSS guide)

http://www.htmlhelp.com/reference/css/ (another CSS guide)

http://www.sitepoint.com/article/css-5-building-skeleton/2 (HTML Utopia: designing without tables [free chapter])

http://en.wikipedia.org/wiki/Cascading_Style_Sheets (It's Wikipedia man!)

http://www.dezwozhere.com/links.html (tons more resources)

http://www.mezzoblue.com/zengarden/resources/ (even more resources [may be some duplicates])

http://www.positioniseverything.net/

Creating CSS buttons vs Image buttons

There have been requests from many community members about the flexibility of the image buttons used in the Magento frontend default theme. This article serves to inform and inspire you to take advantage of what a little tweak in the HTML and CSS can do in swapping image buttons to CSS-powered ones. To those dedicated CSS ladies and gents - **here's to you**.

Let's start by downloading some images

The following gifs are also available in PSD format for your toying pleasure - save it as png, gif, jpg, whatever suits your fancy. For demonstration purposes, we're using the GIF format.

Download the following images, or save the above PSD in your preferred file format, and save it in your magento directory **skin/frontend/default/default/images**

Now for the Magic.

The HTML

Let's use the "Proceed to Checkout" image button in the shopping cart for the occasion (shopping cart phtml can be found in app/design/frontend/default/default/template/checkout/onepage/link.phtml in the Magento file structure)

HTML for Image button - comes with the default Magento package

```
<a href="<?=$this->getCheckoutUrl()?>"><img src="<?=$this-
>getSkinUrl('images/btn_proceed_to_checkout.gif')?>" alt="<?=__('Proceed to Checkout')?>"/>
</a>
```

HTML for CSS button - change the style of the button with just a swap of the background

```
<a class="img-btn btn-checkout" href="<?php echo $this->getCheckoutUrl()?>">

<span><?php echo Mage::helper('checkout')->__('Proceed to Checkout');?>



</span>

</a>
```

Swap out the ‘HTML for Image button’ code with the ‘HTML for CSS button’ code. Note: The ‘img-btn’ class in the HTML just serves to unite all the CSS buttons in case you want to synchronize the font colors and sizes..etc.

The CSS

```
.btn-checkout {
   display:block;
   float:right;
   background:transparent url(../images/btn_proceed_to_checkout_rad.gif) no-repeat 100% 0;
   font-size:15px;
   font-weight:bold;
   padding-right:8px;
   }
.btn-checkout, .btn-checkout:hover {
   color:#fef5e5;
   text-decoration:none;
   }
.btn-checkout span {
   display:block;
   padding:0 17px 0 25px;
   background:transparent url(../images/btn_proceed_to_checkout_bg.gif) no-repeat;
   line-height:40px;
   }
```

And there you have it! - An all CSS-powered button.

Sorry, but with v.1.7 this didn’t work. I changed **link.phtml** code to this:

```
<a class="img-btn btn-checkout" href="<?php echo $this->getCheckoutUrl()?>">

<span><?php echo $this->__('Proceed to Checkout')?>

</span>

</a>
```

And then it worked well... :)

Apply on the Place Order button

copy the file app/design/frontend/default/default/template/checkout/onepage/review.phtml in your design path, and replace :

```
<input type="image" src="<?php echo $this->getSkinUrl('images/btn_place_order.gif') ?>"
onclick="review.save();" value="<?php echo $this->__('Place Order') ?>" />
```

to :

```
<a class="img-btn btn-checkout" href="#" onclick="review.save();"><span><?php echo $this->__('Place Order')
;?></span></a>
```

The “Place Order” button is now css powered and can be translated easily!

Add Home Link with functional active state to Menu Bar (Alternative Method)

UPDATED: Included the following alternative method to add a home link with functional active state to the menu bar.

Add Home Link with functional active state to Menu Bar (Alternative Method)

Find the file called *top.phtml* in *app/design/frontend/default/yourtheme/template/catalog/navigation/* and make the following change:

```
<div class="header-nav-container">

  <div class="header-nav">

    <h4 class="no-display"><?php echo $this->__('Category Navigation:') ?></h4>

    <ul id="nav">



    <!-- ALTERNATIVE HOME BUTTON HACK -->

    <li class="home"><a href="<?php echo $this->getUrl('')?>"><?php echo $this->__('Home') ?></a></li>

    <!-- ALTERNATIVE HOME BUTTON HACK -->



    <?php foreach ($this->getStoreCategories() as $_category):?>

      <?php echo $this->drawItem($_category) ?>

    <?php endforeach ?>

    </ul>

  </div>

  <?php echo $this->getChildHtml('topLeftLinks') ?>

</div>
```

Add the following to the *menu.css* file in *skin/frontend/default/yourtheme/css/*. This example is for a CMS home page which uses *cms-home* class in its body tag.

```
body.cms-home #nav li.home a { color:#d96708; }
```

NOTE: The following comments refer to the following methods.

:::: TRIED THIS AND IT ADDS THE LINK, BUT THE ACTIVE STATE DOES NOT FUNCTION CORRECTLY ::::

Special Note to the person(s) responsible for this page. Myself and many others have attempted to use this code snippet to get the result explained with an active state, but have been unable to get the active state to work. The link does work as far as pointing to exsisting cms page or external pages, but the “active” state does not work at all. Page refreshes and all links adjust back to original state.

FOR HOME LINK ONLY ACTIVE STATE I use the following as it’s the simplist way to get the active state working correctly on the home link.

```
<div class="header-nav-container">

  <div class="header-nav">

    <h4 class="no-display"><?php echo $this->__('Category Navigation:') ?></h4>

    <ul id="nav">



    <!-- WORKING ACTIVE STATE HOME BUTTON HACK -->

    <li class="home<?php if (Mage::helper('core/url')->getCurrentUrl() === Mage::helper('core/url')-
>getHomeUrl()):?> active<?php endif;?>"><a href="<?php echo $this->getUrl('')?>"><?php echo $this-
>__('Home') ?></a></li>

    <!-- WORKING ACTIVE STATE HOME BUTTON HACK -->



    <?php foreach ($this->getStoreCategories() as $_category):?>

      <?php echo $this->drawItem($_category) ?>

    <?php endforeach ?>

    </ul>

  </div>

  <?php echo $this->getChildHtml('topLeftLinks') ?>

</div>
```

Add Home Link to Menu Bar

For the *Home* link in the menu bar of the main template you can add some code to one of the template files.

Find the file called *top.phtml* in *app/design/frontend/default/default/template/catalog/navigation/* and make the following change:

```
<div class="header-nav-container">

  <div class="header-nav">

    <h4 class="no-display"><?php echo $this->__('Category Navigation:') ?></h4>

    <ul id="nav">



    <!-- HOME BUTTON HACK -->

    <?php $_anyActive = false; foreach ($this->getStoreCategories() as $_category) { $_anyActive =
$_anyActive || $this->isCategoryActive($_category); } ?>

    <li class="<?php echo !$_anyActive ? 'active' : '' ?>"><a href="<?php echo $this->getUrl('')?>"><?php echo
$this->__('Home') ?></a></li>

    <!-- HOME BUTTON HACK -->



    <?php foreach ($this->getStoreCategories(10) as $_category):?>

      <?php echo $this->drawItem($_category) ?>

    <?php endforeach ?>

    </ul>

  </div>

  <?php echo $this->getChildHtml('topLeftLinks') ?>

</div>
```

x:x:x: Tried several on this page. The one directly above worked best. I modified it a little bit to contain a link title. Just paste

```
<li class="home<?php if (Mage::helper('core/url')->getCurrentUrl() === Mage::helper('core/url')->getHomeUrl()):?>
active<?php endif;?>"><a href="<?php echo $this->getUrl('')?>" title="Home">Home</a></li>
```

just before

```
<?php echo $_menu; ?>
```

if you're using the default theme or a top.phtml that is based on it. :x:x:x:

Add Home Link to Top Links

The correct way to do this is to open the theme/layout/customer.xml file and then modify the section that shows customer links on all pages, to include a link home and also a link to other customer service pages that you have deemed necessary, e.g. 'returns' (if you get a lot of those enquiries...).

By way of example, this modified XML file includes a 'Home' link and 'Deliveries', 'Returns' and 'Contact Us':

```
<!--
Default layout, loads most of the pages
-->

  <default>

    <!-- Mage_Customer -->
    <reference name="top.links">
      <action method="addLink" translate="label title"
module="customer"><label>Home</label><url></url><title>Home</title><prepare>true</prepare><urlParams/><
position>5</position></action>
      <action method="addLink" translate="label title" module="customer"><label>My Account</label><url
helper="customer/getAccountUrl"/><title>My
Account</title><prepare/><urlParams/><position>94</position></action>
      <action method="addLink" translate="label title"
module="customer"><label>Deliveries</label><url>deliveries</url><title>Deliveries</title><prepare>true</prepar
e><urlParams/><position>95</position></action>
      <action method="addLink" translate="label title"
module="customer"><label>Returns</label><url>returns</url><title>Returns</title><prepare>true</prepare><url
Params/><position>96</position></action>
      <action method="addLink" translate="label title" module="customer"><label>Contact
Us</label><url>contacts</url><title>Contact
Us</title><prepare>true</prepare><urlParams/><position>97</position></action>
    </reference>
  </default>
```

Add Home Link to Top Links - Alternative Method

This will allow you to add a *Home* link in the Top Links (My Account | My Wishlist | Etc.) before the My Account.

Find the file called *links.phtml* in *app/design/frontend/default/default/template/page/template/* and make the following change:

```
<?php $_links = $this->getLinks(); ?>

<?php if(count($_links)>0): ?>

  <div>
```

```
    <ul<?php if($this->getName()): ?>:?> id="<?php echo $this->getName() ?>"<?php endif;?>>



    <!-- HOME BUTTON HACK -->

    <li class="first"><a href="<?php echo $this->getUrl('')?>"><?php echo $this->__('Home') ?></a></li>

    <!-- HOME BUTTON HACK -->



        <?php foreach($_links as $_link): ?>

            <li <?php if($_link->getIsLast()): ?> class="last"<?php endif; ?><?php echo $_link->getLiParams()
?>><?php echo $_link->getBeforeText() ?><a href="<?php echo $_link->getUrl() ?>" title="<?php echo $_link-
>getTitle() ?>" <?php echo $_link->getAParams() ?>><?php echo $_link->getLabel() ?></a><?php echo $_link-
>getAfterText() ?></li>

        <?php endforeach; ?>

    </ul>

  </div>

<?php endif; ?>
```

NOTE: The if statement in the foreach loop has changed since “Home” will always be first, it wasn’t needed. Also note that the <li> for the “Home” link automatically gets the class “first”.

This allows for space between Home | My Account

Embedding HTML in the Footer

http://www.magentocommerce.com/blog/comments/from-the-support-team-embedding-html-in-the-footer/

Often online merchants need to embed code in the site footer to integrate with 3rd party services such as analytics, affiliate programs, tracking codes, etc. Instead of modifying each template manually, Magento offers a quick and simple way to introduce such 3rd party javascript code to the templates directly from the admin interface.

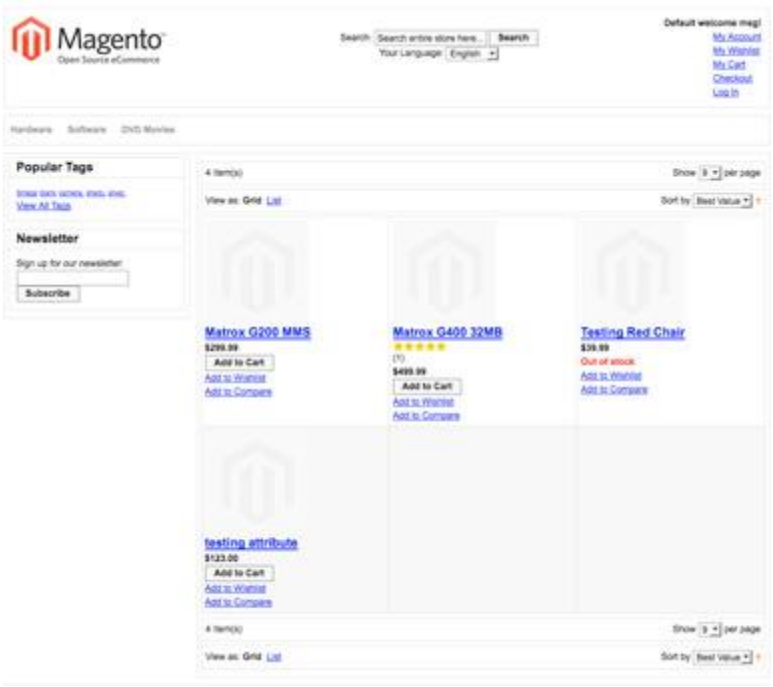
Figure 1

To do so, locate the Miscellaneous HTML box under System>Configuration>General>Design>Footer. Paste the code directly in the box and save the data. Make sure to refresh Magento's cache in full under System>Cache Management.

Once completed, the embedded code will be available on your store.

Blank Theme available through MagentoConnect

http://www.magentocommerce.com/blog/comments/blank-theme-available-through-magentoconnect/



The 'Blank Theme' for Magento is available via MagentoConnect. With 'Blank Theme' the Magento team answers the call of numerous community members out there who's been inquiring about an extra light theme to use as a basis for theme customization. With this theme we've removed all the custom CSS formerly used in Default and Modern Theme, and replaced them with minor typography and positioning CSS assignments. References to the class names were left in the CSS files to provide a solid starting ground for designers/developers. Markups were also revisited and cleaned up in theme-level.

Download, customize, and enjoy the new ultra light Magento theme!

-End of Doc

Thanks from TheUnical Technologies