Joomla Template Tutorial

Last Updated Friday, 04 August 2006

In this tutorial, we'll go through the steps of creating a joomla template. Joomla is an open source Content Management System (CMS) that is freely (literally) available and supported by a large on-line community. Specifically, we will create a template that uses cascading style sheets (CSS) to produce a layout without use of tables. This is a desirable goal as it means that the template code is easier to validate to World Wide Web Consortium (W3C) standards. It will also tend to load faster, be easier to maintain and perform better in search engines. We will discuss these issues in detail later in the tutorial.
Included in this tutorial are the following chapters

- What is a Joomla Template?
Explains what functions are performed by a Joomla template and the difference between a template with no content and when content is added into the CMS

- Localhost Design Process

How the design process differs to that when designing a static (X)HTML web page.

- W3C and Tableless Design

The implications of tableless design in Joomla and the relationship between W3C standards, usability and accessibility

- The Template Components

What files make up a Joomla template and what functions they perform

- Using CSS to create a layout

How to create a source ordered 3 column layout using CSS rather than tables

- The Default Joomla CSS

An introduction to basic CSS styles that should be used with Joomla, along with the default list of styles that are used by the Joomla core

- Modules

How to place, and style modules, including new techniques for rounded corners.

- Menus

A simple strategy to produce lean CSS menus that mimic the effect of those developed with Javascript

- Hiding Columns

How to control when columns are shown and how to hide them when no content is present

- Conclusion

- Appendix A:Tips and Tricks

Variable Page Widths, Rounded Corners, Text Resizers and More

{mospagebreak title=What is a Joomla Template?}
What is a Joomla Template?

The Joomla template is a series of files within the Joomla CMS that

control the presentation of the content. The Joomla template is not a web site, neither is it to be considered a complete web site design. The template is the basic foundation design for viewing your Joomla! web site. To produce the effect of a "complete" web site, the template works hand in hand with the content stored in the Joomla databases. An example of this can be seen below:

This screenshot shows the template in use with sample content. Figure B shows the template as it may look with a raw Joomla installation with little or no content. The template is styled so that when your content is inserted, will inherit the stylesheet defined in the template such as link styles, menus, navigation, text size and colors to name a few. Notice how the images associated with the content (the photos of the people) are not part of the template, whereas the image in the header is part of the template.

Using a template for a CMS, as Joomla does, has a number of advantages and disadvantages:

- There is a complete separation of content and presentation, especially when CSS is used for layout (as opposed to having tables in the index.php file). This is one of the main criteria for a site that meets modern web standards.

- A new template, and hence a completely new look to a web site can be applied instantly. This can even have different locations/positioning of content as well as colors and graphics.

- If different layouts are called for within one web site, it is difficult to achieve. Although different templates can be applied to different pages, this built in functionality is not reliable. Most designers choose to use various PHP code to show/hide columns depending if there is any content published in that location (discussed in tips and tricks).

{mospagebreak title=Localhost Design Process}

Localhost Design Process

The web page you see at a Joomla powered web site is not static. That means it is generating dynamically from content stored in the database. The page that you see is created through various PHP commands that are in the template. This presents some difficulties in the design phase.

It's common now to use a "what you see is what you get" (WYSIWYG) HTML editor such as Dreamweaver. This means that the designer does not even need to code the HTML. However, this is not possible in the Joomla template design process, WYSIWYG editors cannot display a dynamic page. This means that the designer must code "by hand" and the view the output page from the PHP on a served page. With a fast enough connection this could be a web server, but most designers use a "local server" on their own computer. This is a piece of software that will serve the web pages on the designer's computer.

There is no "right way" to create a web page, it depends on the designer's own background. Those more graphical make an "image" of a page in a graphics program like Photoshop and then break up the images to use (known as slice and dice). More technologically inclined designers will often just jump straight into the CSS and start coding.

However, as mentioned above, the Joomla template designer is limited in that he cannot instantly see the effect of his/her coding in the same editor, the modified design process is:

- Make edits with HTML editor, save changes

- Have localhost server running in background to "run" Joomla.

- View edits in a web browser

- Go to 1.

Localhost Server Options

Several local host servers are available:

- JSAS - Joomla Stand Alone Server. WAMP server that serves up Joomla on Windows based PCs. A self contained Apache - MySQL - PHP server

  jsas.joomlasolutions.com

- XAMPP is an easy to install Apache Distribution for Linux, Windows, Mac OS X, and Solaris. The package includes the Apache web server, MySQL, SQLite, PHP, etc

  www.apachefriends.org/en/xampp.html

JSAS does have a significant amount of advertising on it should that concern you.

Easy CSS Styling

One useful technique to make the design process more efficient is to serve a web page that you are designing and then copy and paste the source into an editor. For example, once your layout CSS is set up, you can use one of these localhost servers to serve a page, then View_Source. You then copy and paste that into your editor. You can now easily style the page using CSS and not have to go through the cycle of steps described above.

Some HTML Editor Options for Joomla Designers

For those not able to pay for a commercial editor such as Dreamweaver, there a some free editors available.

Nvu is a solid choice and has built in validation. It also a Mambo/Joomla extension which will be helpful. Nvu is 100% open source. This means anyone is welcome to download Nvu at no charge, including the source code if you need to make special changes. You can get the extension from Mamboforge. The complete forge project is here.

Another choice is actually more of a validator. The "CSE HTML Validator" is an all-in-one HTML, XHTML, CSS, link, spelling, and accessibility checker. You can get the free version html validator here.

Note that neither of these are "WYSIWYG" html editors.

{mospagebreak title=W3C and Tableless Design}
W3C and Tableless Design

Usability, accessibility and search engine optimization are all phrases used to describe high quality web pages in today's world wide web. In reality, there is a significant amount of overlap between them and a web page that demonstrates the characteristics of one does so for all three. The easiest way to achieve these three goals is to do so using the framework laid out in the W3C web standards.

For example, a site that is (x)html semantically structured (the xhtml explains the document, not how it looks) will be easily read in a screen reader by someone who has poor vision. It will also be easily read by a search engine spider. Google is effectively blind in how it reads your web site

A site that is validates to the World Wide Web Consortium's (W3C) web standards has a much better foundation for making it accessible, usable and search engine optimized. Think of these as building codes for your house. A web site built with them is stronger and safer. You can check your pages with the W3C's HTML validation service. for free. At its simplest, a site that meets W3C validation uses semantic (x)html and separates content from presentation using CSS.

To help you understand where web standards came from, some history is helpful. Many web pages are actually designed for older browsers. Why? Browsers have continually evolved since the www started. New ones have appeared and old ones have disappeared (remember Netscape?). Another complicating factor is that different browser makers (like Microsoft) tend to have their browsers interpret html/xhtml in slightly different ways. This has lead to web designers having to design their websites to support older browsers rather than new ones. It's often decided that the web page needs to appear properly to these "legacy" browsers.

Web standards put into place a common set of "rules" for all web browsers to use to show a web page. The main organization pushing these standards is the World Wide Web Consortium (WC3), whose Director, Tim Berners-Lee has the distinction of actually inventing the world wide web in 1989.

Ask five designers what web standards are and you will get five answers. But most agree that they are based on the following:

- Valid code, whether html or xhtml (or others)

Earlier we used an example of building codes for construction. The standards outlined for the code that makes a web page have been developed to achieve consistency. It's easy to check your code at validator.w3.org. Make sure you use the correct DOCTYPE when you try and validate your code. This article at Compass Design describes a valid Joomla doctype.

- Semantically Correct Code

We mentioned before that being semantic means that the (x)html in the web page describes only content, not presentation. In particular this means structured organization of h1/h2 etc tags and only using tables for tabular data, not to layout a web page.

- Cascading Style Sheets (CSS)

Closely related to having semantic code, is using cascading style sheets to control the look and layout of your web page. Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents. (Source: www.w3.org/Style/CSS/). They exist parallel to the (x)html code and so let you completely separate content (semantic code) form presentation (CSS). The best example of this is CSS Zen Garden, a site where the same semantic xhtml is shaped in different and unique ways with different CSS. The result is pages that look very different but have the same core content.

Designing Joomla powered sites currently presents considerable challenges to meet validation standards. In the current series of releases, 1.0.X, the code uses a significant amount of tables to output its pages. This isn't really using CSS for presentation, nor does it produce semantically correct code. This problem is compounded by the fact that very few 3rd party developers are using CSS either, most use table to generate their code too. However, tableless is not the same as valid. Its quite possible to have a site that uses tables to validate, it just makes it harder. A useful thread on the Joomla forums go into this in more detail:

Easy tricks to remove many tables from the standard output of Joomla!

Fortunately, the Joomla Core Development team recognize this issue of Joomla. While in 1.5 there will be no changes towards removing tables from the core, a roadmap has be defined that begins to address this in the 1.6 release and on.

Regardless, care can still be taken when creating a template so that it is accessible (e.g. scalable font sizes), usable (e.g. clear navigation) and optimized for search engines (e.g. source ordered).
{mospagebreak title=The Template Components}
The Template Components

In order to understand the contents of a template, we will start by looking at a blank joomla template. In this file are the various files and folders that make up a Joomla template. These files must be placed in the /templates directory of a Joomla installation. So, if we had two templates installed, our directory would look something like:

/templates/JS_Smoothportal

/templates/JS_Synergy

Note that the directory names for the templates must be the same as the name of the template, in this case JS_Smoothportal and JS_Synergy.
Obviously they are case sensitive and shouldn't contain spaces.
Traditionally the designers initials or name is used as a prefix.

Within the directory of a template, there are a number of key files:

/JS_Smoothportal/templateDetails.xml

/JS_Smoothportal/index.php

These two filenames and location must be matched exactly as this is how they are called by the Joomla core script.

- templateDetails.xml
(note the
uppercase "D") An XML format metadata file that tells Joomla! what other files are needed when loading a web page that uses this template. It also details the author, copyright and what files make up the template (including any images used). The last use of this file is for installing a template when using the admin backend.

- index.php

This file is the most important. It lays out the site and tells the joomla CMS where to put the different components and modules. It is a combination of PHP and (X)HTML.

In almost all templates, additional files are used. It is conventional (although not required by the core) to name and locate them as shown below:
/JS_Smoothportal/template_thumbnail.png

/JS_Smoothportal/css/template_css.css

/JS_Smoothportal/images/
logo.png

- template_thumbnail.png
A web Browser
screenshot of the template (usually reduced to around 140 pixels wide and 90 pixels high). After the template has been installed, this functions as a "Preview image" visible in the Joomla! administration Template Manager.

- css/template_css.css

The CSS of the template. The folder location is optional, but you have to specify where it is. Note that the file name is only important in that its referenced in index.php. You could call it what you like. Usually the name shown is used, but we will see later that there are advantages in having other css files too.

- images/logo.png

Any images that go with the template. Again for organization reasons, most designers put this in an images folder. Here we have a image file called logo.png as an example.

To add the template (again, copious tutorials exist) you go to the admin portion of your site and install the template by uploading the zip file. Note you can actually add the files individually (not in a zip) too. You have to put them in yoursite.com/templates.
templateDetails.xml

The templateDetails.xml must include all the files that are part of the template. It also includes information such as the author and copyright. Some of these are shown in the admin backend in the Template Manager

An example xml file is shown below:

```
<mosinstall type="template" version="1.0.x">

  <name>YourTemplate</name>

  <creationDate>March 06</creationDate>

  <author>Barrie North</author>

  <copyright>GNU/GPL</copyright>

  <authorEmail>compassdesigns@gmail.com</authorEmail>
  <authorUrl>www.compassdesigns.net</authorUrl>

  <version>1.0</version>

  <description>
  An example template that shows a basic xml details file   </description>

  <files>

   <filename>index.php</filename>

   <filename>js/ie.js</filename>

   <filename>template_thumbnail.png</filename>

  </files>

  <images>

   <filename>images/header.png</filename>

   <filename>images/background.png</filename>

   <filename>template_thumbnail.png</filename>

  </images>

  <css>
   <filename>css/base.css</filename>

   <filename>css/norightcol.css</filename>
```

```
    <filename>css/template_css.css</filename>

  </css>

  </mosinstall>
```

Lets explain what some of these lines mean:

- mosinstall

The contents of the XML document are instructions for the installer. the option type="template" tells the installer that we are installing a template

- name:

Defines the name of your template. The name you enter here will also be used to create the directory within the templates directory. Therefore it should not contain any characters that the file system cannot handle, for example spaces. If installing manually, you need to create a directory that is identical to the template name.

- creationDate:

The date the template was created. It is a free form field and can be anything like May 2005, 08-June-1978, 01/01/2004 etc.

- author:

The name of the author of this template - most likely your name

- copyright:

Any copyright information goes into this element. A Licensing Primer for Developers & Designers can be found on the Joomla forums.

- authorEmail:

Email address where the author of this template can be reached.

- authorURL:

The URL of the author's web site

- version:

The version of this template

- files:

The "files" sections contains all generic files like the PHP source for the template or the thumbnail image for the template preview. Each file listed in this section is enclosed by <filename> </filename>. Also included would be any additional files, here we use the example of a JavaScript file that is required by the template.

- images:

All image files that the template uses are listed in the images section. Again each file listed is enclosed by <filename> </filename>. Path information for the files is relative to the

root of your template, e.g. if your template is in the directory called 'YourTemplate' and all images are in a directory 'images' that is inside 'YourTemplate', the correct path is:

<filename>images/my_image.jpg</filename>

- css:

The stylesheet is listed in the css section. Again the filename is enclosed by <filename> </filename> and it's path is relative to the template root. Its often useful to have a number of stylesheets used all imported by the main template_css.css. We will discuss that more later in the tutorial.

The index.php

What actually is in an index.php file? It is a combination of (X)HTML and PHP that determines everything about the layout and presentation of the pages.

First we will look at a critical part of achieving valid templates, the DOCTYPE at the top of the index.php file. This bit of code that code goes at the very top of any web page. At the top of our page we have this in our template:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

lang="<?php echo _LANGUAGE; ?>"
xml:lang="<?php echo _LANGUAGE; ?>">
```

A web page DOCTYPE is part of the fundamental components of who a web page is shown by a browser, specifically, how that browser interprets CSS. To give you a sense, an observation from alistapart.com says:

[information on W3C's site about doctypes is] "written by geeks for geeks. And when I say geeks, I don’t mean ordinary web professionals like you and me. I mean geeks who make the rest of us look like Grandma on the first day She’s Got Mail.™"

Anyway, there are several doctypes you can use. Basically, the doctype tells the browser how to interpret the page. Here the words "strict" and "transitional" start getting floated around (float:left and float:right usually). Essentially, ever since the WWW started, different browsers have had different levels of support for CSS. This means for example, that Internet Explorer won't understand the "min-width" command to set a minimum page width. To duplicate the effect you have to use "hacks" in the CSS.

Strict means the html (or xhtml) will be interpreted exactly as dictated by standards. A transitional doctype means that the page will be allowed a few agreed upon differences to the standards.

To complicate things, there is something called "quirks" mode. If the doctype is wrong, outdated, or not there, then the browser goes

into quirks mode. This is an attempt to be backwards compatible, so Internet Explorer for example, will render the page pretending as if it was IE4.

Unfortunately, people sometimes end up in quirks mode accidentally. It usually happens two ways:

- They use the doctype declaration straight from the WC3 web page, the link ends up as:

DTD/xhtml1-strict.dtd
Except this is a relative link on the WC3 server. You need the full path as shown above.

- Microsoft set up IE6 so you could have valid pages, but be in quirks mode. This happens by having an "xml prolog" put before the doctype.

<?xml version="1.0" encoding="iso-8859-1"?>

The part about IE6 quirks mode is important. In this tutorial we will only be designing for IE6+, so we will make sure that its running in standards mode. This will minimize the hacks we have to do later on. The xml prolog isn't essential anyway, we'll be taking note of future releases of Joomla and be leaving it off.

Making a page standards compliant, where you see "valid xhtml" at the bottom of the page does not mean really difficult coding, or hard to understand tags. It merely means that the code you use matches the doctype you said it would. That's it! Nothing else. Designing your site to standards can on one level be reduced to saying what you do, and then doing what you say.

Some useful links:

- http://www.quirksmode.org/css/quirksmode.html

- http://www.alistapart.com/stories/doctype

- http://www.w3.org/QA/2002/04/Web-Quality

- http://forum.joomla.org/index.php/topic,7537.0.html

- http://forum.joomla.org/index.php/topic,6048.0.html

What else is in index.php?

Let's look at the structure of the header first, we want to be as minimal as possible, but still have enough for a production site. The header information we will use is:

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="<?php echo _LANGUAGE; ?>"
```

```
xml:lang="<?php echo _LANGUAGE; ?>"

<head>


<meta http-equiv="Content-Type" content="text/html; <?php echo _ISO; ?>" />

<?php
if ($my->id) {
initEditor();
}
?>


<?php mosShowHead(); ?>


<script type="text/javascript"> </script>

<!--http://www.bluerobot.com/web/css/fouc.asp-->


<link href="templates/<?php echo $cur_template; ?>/css/template_css.css" rel="stylesheet"
 type="text/css" media="screen" />


</head>
```

What does all that mean?

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>
```

Makes sure that the file isn't being accessed directly.

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="<?php echo _LANGUAGE; ?>"
xml:lang="<?php echo _LANGUAGE; ?>"

<head>
```

We talked about this above. The "<?php echo _LANGUAGE; ?>" is pulling the language from the site global configuration.

```
 <meta http-equiv="Content-Type" content="text/html; <?php echo _ISO; ?>" />
```

What character set we are using, _ISO is a special constant defining the character set encoding to use.

```
<?php
  if ($my->id) {
  initEditor();
  }
 ?>
```

This is a script variable that is non-zero if a user is logged in to your site. If a user is logged in then the nominated WYSIWYG Editor is pre-loaded. You may, if you wish, always pre-load the Editor, but generally an anonymous visitor will not have the need to add content. This saves a little script overhead for normal browsing of your site.

```
<?php mosShowHead(); ?>
```

Header stuff that is set in the global configuration again. It includes the following tags (in a default installation):

```
  <title>A Complete Guide to Creating a Joomla Template </title>
  <meta name="description" content="Installing Joomla, doctype and the blank joomla template" />
  <meta name="keywords" content="installing joomla, joomla doctype, blank joomla tempate" />
  <meta name="Generator" content="Joomla! - Copyright (C) 2005 Open Source Matters. All rights reserved." />
  <meta name="robots" content="index, follow" />
  <link rel="shortcut icon" xhref="images/favicon.ico" />

<script type="text/javascript"> </script>
```

To stop a bug, that being a flash of un-styled content. Details courtesy of Blue Robot. Note this can be any script file, so if we add one, we can remove this line.

```
<link href="templates/<?php echo $cur_template; ?>/css/template_css.css"
 rel="stylesheet" type="text/css" media="screen" />
```

This line links to the CSS file for the template. The PHP code <?php echo $cur_template; ?> will return the name of the current template. This makes this line "portable". When you create a new template you can jsut copy it (along with the whole head code) and not worry about editing anything.

As you will see later, in the temmplate_css.css file, we will use @import as a way to stop the site breaking with Netscape 4. Users of very old browsers won't be able to get the CSS sheet so will see our neat un styled content. If you wanted to cater to these older browsers, we would have too many CSS hacks, so we do this.
A blank joomla template body

This will be very very easy! Ready?

```
<body>

  <!-- 1 --><?php echo $mosConfig_sitename;?>

  <!-- 2 --><?php mospathway()?>

  <!-- 3 --><?php mosLoadModules('top');?>

<!-- 4 --><?php mosLoadModules('left');?>

<!-- 5 --><?php mosMainBody();?>
```

```
<!-- 6 --><?php mosLoadModules('right');?>


<!-- 7 --><?php include_once( $mosConfig_absolute_path .'/includes/footer.php' );?>

</body>
</html>
```

We have in a reasonably logical order:

- name of the site
- the pathway
- top module
- left modules
- main content
- right modules
- the default footer module

The goal is to try and come as close to semantic markup as possible. From a web point of view, it means a page can be read by anyone, a browser, a spider or a screen reader. Semantic layout is the cornerstone of accessibility.

Now its worth noting that what we have here really is only the potential for semantic layout. If one were to go ahead and put random modules in random locations, then you would have a mess. An important consideration for CMS sites, a template is only as good as the population of the content. It is this that often trips designers up when trying to validate their site.
{mospagebreak title=Using CSS to create a layout}
Using CSS to create a layout

We will be using CSS to make a 3 column layout for the Joomla template. We will also be making it a fluid layout. There are two main types of web page layout, fixed and fluid, and they both refer to how the width of the page is controlled.

The width of the page is an issue because of the many browser resolutions that people surf the web at. Although the percentage is dropping, about 20% of surfers are using an 800x600 resolution. The majority, 76%, are using 1024x768 and higher (source:www.upsdell.com). Making a fluid layout means that your valuable web page won't be a narrow column in the 1024 resolution, and will all be visible on smaller monitors.

A typical design might use tables to layout the page. They are useful in that you just have to set the width of the columns as percentages, but they have several drawbacks:

- They have lots of extra code compared to CSS layouts. This leads to longer load times (which surfers don't like) and poorer performance in search engines. The code can roughly double in size, not just with markup but also something called "spacer gifs". Even big companies sometimes fall into the table trap as seen by a recent contorversey about the new disney.co.uk website.

- They are difficult to maintain. To change something you have to figure out what all the td/tr are doing. With CSS there are just a few lines to inspect.

- The content cannot be source ordered. Many surfers of the web do not see web pages on a browser. Those viewing with a text browser or screen reader will read the page from the top left corner to the bottom right. This means that they first view everything in the header and left column (for a 3 column layout) before they get the the middle column, the important stuff. A CSS layout on the other hand allows for "source-ordered" content, which means the content can be rearranged in the code/source. Perhaps your most important site visitor is Google, and it uses a screen reader for all intents and purposes.

Let's look at our layout using CSS. You can position elements (stuff) in several ways using CSS. For a quick introduction a good source is Brainjar's CSS Positioning.

If you are new to CSS you might read at least one "beginners guide to CSS". Here are a few suggestions:

Kevin Hale's - An Overview of Current CSS Layout Techniques

htmldog's CSS Beginner's Guide

Mulder's Stylesheets Tutorial

yourhtmlsource.com

We will be using float to position our content. At its most basic, the template might look like this:

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="<?php echo _LANGUAGE; ?>"
xml:lang="<?php echo _LANGUAGE; ?>"

<head>

<meta http-equiv="Content-Type" content="text/html; <?php echo _ISO; ?>" />

<?php

if ($my->id) {
initEditor();
}
?>

<?php mosShowHead(); ?>

<script type="text/javascript"> </script>

<!--http://www.bluerobot.com/web/css/fouc.asp-->
```

```
<link xhref="templates/<?php echo $cur_template; ?>/css/template_css.css" rel="stylesheet"
 type="text/css" media="screen" />

<style type="text/css">
 <!--


 #wrap {width:80%;}


 #header {}


 #sidebar {float:left;width:20%;}


 #content {float:left;width:60%;}


 #sidebar-2 {float:left;width:20%;}


 #footer {clear:both;}


 -->
 </style>


 </head>


 <body>


<div id="wrap">

<div id="header">


   <?php echo $mosConfig_sitename; ?>
   <?php mospathway() ?>

   </div>


<div id="sidebar">


   <?php mosLoadModules('left');?>


   </div>


<div id="content">
 <?php mosLoadModules('top');?>
   <?php mosMainBody(); ?>


   </div>
```

```
<div id="sidebar-2">

  <?php mosLoadModules('right');?>

  </div>

<div id="footer">

 <?php include_once( $mosConfig_absolute_path .'/includes/footer.php');?>

  </div>



 </div> <!--end of wrap-->

</body>
</html>
```

The CSS styles are defined here in the head of the file to show what is going on, but normally they would be in the template_css.css file.

Everything is contained in a box/element called #wrap. This had a width of 80% of the viewport at any time.

CSS Shorthand

Its possible to reduce the amount of CSS by using "shorthand". One example of this is padding and margin styles applied to an element.

```
margin-top:5px;
margin-bottom:5px;
margin-left:10px;
margin-right:10px;
```

can be replaced by:

```
margin: 5px 10px;
```

There are 'shorthand' styles at the beginning of each style definition. Once you have figured out the styles, fill the shorthand versions in and delete the long versions. The syntax is:
font: font-size |font-style | font-variant | font-weight | line-height | font-family

Here is an example, rather than this...

```
font-size:1em;
font-family:Arial,Helvetica,sans-serif;
font-style:italic;
font-weight:bold;
line-height:1.3em;
```

Have this:

```
font:bold 1em/1.3em Arial,Helvetica,sans-serif italic;
```

Read more at An Introduction to CSS shorthand properties about this syntax.

The left, middle and right columns are each given their own element. Each is floated left and given a percent width that add up to 100%. The clear:both
style on the footer tells the browser to "stop floating" and makes the footer stretch across all three columns.

To improve the layout, and to add some breathing room to the content, we need to add some column spacing, commonly called "gutter".
Unfortunately, there is a problem here. You might know that Internet Explorer does not interpret CSS correctly. One problem is that calculates width differently. We solve this problem by not using any padding or borders on something that has a width. To get our gutter we add another element inside the columns. This is shown below:

```
<div id="sidebar">

  <div class="inside">

  <?php mosLoadModules('left');?>

  </div></div>

<div id="content">

  <div class="inside">
 <?php mosLoadModules('top');?>

   <?php mosMainBody(); ?>

   </div></div>

<div id="sidebar-2">

<div class="inside">

<?php mosLoadModules('right');?>

</div></div>
```

To the CSS we add:

```
.inside {padding:10px;}
```

This simple layout is a good one to use for learning about how to use CSS with Joomla. It gives two of the advantages of CSS over table based layouts, it is less code and is easier to maintain. However, it is not source ordered. For that we must use a more advanced layout known as a "nested float". With his kind permission, we will be adapting a layout developed by Dan Cederholm and described in more detail in his book.
Source Ordered Three Column CSS Layout

To help explain how we are doing this, let's look at the end result first.

[TO DO: PICTURE OF NESTED FLOAT HERE]

The page is split into two main floats. The first, #main-body is floated left, the second, #sidebar-2 is floated right. This is the same as we did before, the #main-body float will appear first in the source code. Now, within main-body, we have two more floats; #content is floated right and #sidebar is floated left. As long as we set our widths correctly, the #content float can appear first in the source code.

```
<div id="wrap">

<div id="header">

<?php echo $mosConfig_sitename; ?>

<?php mospathway() ?>

</div>

<div id="main-body">

<div id="content">

<div class="inside">

<?php mosLoadModules('top');?>

  <?php mosMainBody(); ?>

  </div></div>

<div id="sidebar">

 <div class="inside">

 <?php mosLoadModules('left');?>

 </div></div>

</div> <!--end of main-body-->

<div id="sidebar-2">

<div class="inside">

<?php mosLoadModules('right');?>
```

```
</div></div>



<div id="footer">
<?php include_once($mosConfig_absolute_path .'/includes/footer.php');?>

</div>


</div> <!--end of wrap-->
```

So now in the source code we have the order of:

```
#content
#sidebar
#sidebar-2
```

To figure out the widths, we now need to do a little math. Let's say we want the side columns to be 25% each. #sidebar-2 is easy, it will just have width:25%. However, #sidebar will need a width defined based on it being within a that has a width of 75%. Its width needs to be 33%.

So, 33% of 75% = 25%

The width of #content will need to be the remainder. We will set it to 66%. The last 1% we split between #content and #sidebar.

The CSS will be:

```
#wrap {width:80%;}
#header {}
#footer {
clear:both;
}
#main-body {
float:left;
width:75%;
}
#sidebar-2 {
float:right;
width:25%;
}
#content {
float:right;
width:66.5%;
}
#sidebar {
float:left;
width:33.5%;
}
.inside {
padding:10px;
}
```

Some CSS designers would recommend building in a small gutter by making the side columns fractionally smaller. This helps the layout

stop from breaking in Internet Exporer. If you wish do do this, simply change the width of #sidebar-2 to 24%

The code of the template is shown below. Its in a scroll box so you can copy and paste into the index.php. Note we have removed the layout CSS from the head. We'll be putting it into a seperate file.

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>

 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

 <html xmlns="http://www.w3.org/1999/xhtml" lang="<?php echo _LANGUAGE; ?>"
  xml:lang="<?php echo _LANGUAGE; ?>"

 <head>

 <meta http-equiv="Content-Type" content="text/html; <?php echo _ISO; ?>" />

 <?php

  if ($my->id) {
  initEditor();
  }
  ?>

 <?php mosShowHead(); ?>

 <script type="text/javascript"> </script>

 <!--http://www.bluerobot.com/web/css/fouc.asp-->

 <link href="templates/<?php echo $cur_template; ?>/css/template_css.css" rel="stylesheet"
  type="text/css" media="screen" />

 </head>



 <body>



<div id="wrap">



<div id="header">

<?php echo $mosConfig_sitename; ?>

<?php mospathway() ?>

</div>



<div id="main-body">



<div id="content">
```

```
<div class="inside">

<?php mosLoadModules('top');?>

<?php mosMainBody(); ?>

</div></div>


<div id="sidebar">

<div class="inside">

<?php mosLoadModules('left');?>

</div></div>


</div> <!--end of main-body-->


<div id="sidebar-2">

<div class="inside">

<?php mosLoadModules('right');?>

</div></div>


<div id="footer">
  <?php include_once($mosConfig_absolute_path .'/includes/footer.php');?>

</div>


</div> <!--end of wrap-->


</body>
  </html>
```

{mostitle=The Default Joomla CSS}
The Default Joomla CSS

So far all of our CSS has been only about layout. That will make a pretty plain page, so let's add some formatting. We will also move all of the CSS code out of the header of the index.php and into CSS files.

Although it might slow your site down by a tiny amount, it is more flexible to have sub-files for CSS, and then have them all imported by the template_css.css, an example might look like this:
/*Compass Design template CSS file*/

```
  @import url("layout.css"); /*layout css file*/

/*
@import url("color.css"); color css file*/


@import url("customize.css"); /*Use this file to customize your website*/
```

As mentioned earlier, we use @import
because Netscape 4 does not understand this command. It also doesn't
undertsand CSS, so it will just see our unstyled content as a text
browser would.

All of the CSS relating to layout would go in the layout.css file. Once set up you can just leave it and know that any changes you make to other stylesheets won't do anything drastic. The color.css
file might contain anything related to color (its commented out as
shown here). You can then quickly and easily make changes or set up
"color packs". Lastly, all our typography and Joomla styling would go
in our customize.css file.

Our layout.css file is now:

```
/*Compass Design layout.css CSS file*/
body {
text-align:center; /*center hack*/
}
#wrap {
width:80%; /*center hack*/
margin:0 auto; /*center hack*/
text-align:left;
}
#header {
text-align:left;
}
#footer {
clear:both;
}
#main-body {
float:left;
width:75%;
}
#sidebar-2 {
float:right;
width:25%;
overflow:hidden;
margin-left:-3px;
}
#content {
float:right;
width:66.5%;
overflow:hidden;
}
#sidebar {
float:left;
width:33.5%;
overflow:hidden;
}
.inside {
padding:10px;
}
```

We have centered the page by using a small hack. This has to be done because of Internet Explorer. With standards compliant browser we could just say margin:0 10%;
to center the page, but IE does not recognize that. So we center the "text" of the whole page and then align it back left in the columns.

We have also added two more rules. One is overflow:hidden to each column. This will make the page "break" more consistantly as we reduce its width. Secondly we have added a negative margin to sidebar-2. This is purely for Internet Explorer to address some of its issues with CSS. We could apply this rule only to IE by adding a hack (the Tan hack):
* html #sidebar-2 {margin-left:-3px;}

However, hacks are generally troublesome. Its better (in this author's opinion) to apply the rule to all browsers, after all, its just 3 pixels.

At the beginning of the customize.css file we will set some overall styles and have what is known as a "global reset".

```
/*Compass Design Customize.css file */

* {
margin:0;

padding:0;

}
h1,h2,h3,h4,h5,h6,p,blockquote,form,label,ul,ol,dl,fieldset,address {
margin: 0.5em 0;
}
li,dd {
margin-left:1em;
}
fieldset {
padding:.5em;
}
body {
font-size:76.1%;
font-family:Verdana, Arial, Helvetica, sans-serif;
line-height:1.3em;
}
#header {
background:#0099FF;
}
#footer {
background:#0099FF;
}
#main-body {
background: #CC0000;
}
#sidebar-2 {
background:#009933;
}
#content {
background: #999999;
}
#sidebar {
background: #009933;
}
```

Everything is given a zero margin and padding and then all block level elements are given a bottom margin. This helps acheive browser consistancy. You can read more about the global reset at clagnut and left-justified.

The font size is set to 76.1%. The reason for this is to try and get more consistant font sizes across browsers. All font sizes are then set in em. Having line-height:1.3em
helps readbility. This means that the pages will be more accessible as the viewer will be able to resize the fonts to their own preference. This is discussed more at:

An experiment in typography at The Noodle Incident (Owen Briggs)

Lastly we have added some background colors so that we can see where the columns are.

With a fresh default installation with Joomla 1.0.8, the template now looks like this:

Notice that the side columns do not reach their footer. This is because they only extend as far as their content, where the space is red on the left and white on the right, they don't exist. If we have a template that has a white background for all three columns, this is no problem. We will use this approach and will have boxes round the modules. If equal height columns are desired that are colored, or have boxes, you must use a background image that will tile vertically. This technique is called "Faux Columns" and is described by Douglas Bowman and Eric Meyer.

[TO DO: DESCRIPTION OF FAUX COLUMNS HERE]

Unfortunately, this technique causes a few problems in Internet Explorer. In some situations, the column background will disappear. This is known as the "Peekaboo bug" and is described in more detail at Position Is Everything. It is fixed by applying the Holly Hack (assigning a height of 1% in IE). Here it is slightly modified so that only IE6 is targeted using an !Important statement. This means that no actual hack, i.e. invalid CSS is used.

```
#wrap{
border:1px solid #999;
background: url(../images/threecol-r.gif) repeat-y 75% 0;
height:100% !Important;height:1%;
}

#wrap-inner {
background: url(../images/threecol-l.gif) repeat-y 25.125% 0;
height:100% !Important;height:1%;
}
```

Note at very small screen widths (<600px) in Internet Explorer, the layout will start breaking. Its possible to fix this by hacking a minimum width, but we will leave that as an exercise for the designer.

Joomla Specific CSS

At the time of writing, the current stable of Joomla is the 1.0.X series. This release still uses significant tables to output content in the main body. Along with these tables there are is CSS output available to the designer to style pages. Based on some research by

various community members, the current list is shown below. Note it does not include generic web pages styles like H1, H2, p, ul, a, form etc.

#active_menu
#blockrandom
#contact_email_copy
#contact_text
#emailForm
#mod_login_password
#mod_login_remember
#mod_login_username
#poll
#search_ordering
#search_searchword
#searchphraseall
#searchphraseany
#searchphraseexact
#voteid1,#voteid2....
.adminform
.article_seperator
.back_button
.blog
.blog_more
.blogsection
.button
.buttonheading
.category
.clr
.componentheading
.contact_email
.content_rating
.content_vote
.contentdescription
.contentheading
.contentpagetitle
.contentpane
.contentpaneopen
.contenttoc
.createdate
.fase4rdf
.footer
.frontpageheader
.inputbox
.latestnews
.mainlevel
.message
.modifydate
.module
.moduletable
.mostread
.newsfeed
.newsfeeddate
.newsfeedheading
.pagenav
.pagenav_next
.pagenav_prev
.pagenavbar
.pagenavcounter
.pathway
.polls
.pollsborder
.pollstableborder
.readon
.search
.searchintro
.sectionentry1

.sectionentry2
.sectionheader
.sitetitle
.small
.smalldark
.sublevel
.syndicate
.syndicate_text
.text_area
.toclink
.weblinks
.wrapper

Important note about this list.

Many designs you might see actually have given CSS styles that are more specific in their definition. Basically, a more specific rule will over ride a less specific rule.

For example:
a {color:blue;}
a:link {color:red;}

.contentheading {color:blue;}
div.contentheading {color:red;}

The color on a link and the color of the .contentheading will be RED, as that rule is more specific (as .contentheading is contained within a)

In the case of Joomla templates, you will often see more specific rules used. This often occurs when the class is on a table. More examples:
.moduletable
table.moduletable

- .moduletable is the name of the that wraps a module. table.moduletable will only apply the style to a table with class="moduletable" on it.

- .moduletable will apply the style regardless of what element the class is on.

a.contentpagetitle:link
.contentpagetitle a:link

- a.contentpagetitle:link will apply the style to any a tags with a .contentpagetitle class on them that is a link.

- .contentpagetitle a:link will apply the style to any elements INSIDE .contentpagetitle that are a link.

Specificity is not easy to understand, its often easier to start by using the most general style possible and then getting more specific if the results are not what you expect.

Some links about specificity:

http://www.htmldog.com/guides/cssadvanced/specificity/

http://www.meyerweb.com/eric/css/link-specificity.html

http://www.stuffandnonsense.co.uk/archives/css_specificity_wars.html

At the moment our template is using alot of tables, 20 in fact! As
mentioned earlier, this slows the pages down and makes them harder to
update. To reduce the number of tables we need to use $style suffixes in the index.php when we call the modules.
{mospagebreak title=Modules}
Modules

When a module is called in the index.php, it has several option on how it is displayed. The syntax is:
mosLoadModules('$position_name'[, $style])

The $style is optional and can be 0, 1, -1, -2 or -3.

0 = (default display) Modules are displayed in a column. The following shows an example of the output:
<table cellpadding="0" cellspacing="0" class="moduletable">

<tr>

<th valign="top">Module Title</th>

</tr>

<tr>

<td>Module output</td>

</tr>

</table>

1 = Modules are displayed horizontally. Each module is output in the
cell of a wrapper table. The following shows an example of the output:
<!-- Module wrapper -->

<table cellspacing="1" cellpadding="0" border="0" width="100%">

<tr>

<td align="top">
<table cellpadding="0" cellspacing="0" class="moduletable">

<tr>

```
<th valign="top">Module Title</th>

</tr>

<tr>

<td>
Module output</td>

</tr>

</table>

</td>

<td align="top">
<!-- ...the next module... -->

</td>

</tr>

</table>
```

-1 = Modules are displayed as raw output and without titles. The following shows an example of the output:

```
Module Output
```

-2 = Modules are displayed in CSS format enclosed by a . The following shows an example of the output:

```
<div class="moduletable">

  <h3>Module Title</h3>

  Module output

</div>
```

-3 = Modules are displayed in a format that allows, for example, stretchable rounded corners. If this $style is used the name of the changes from moduletable to module. The following shows an example of the output:

```
<div class="module">

  <div>
<div>
<div>
```

```
<h3>Module Title</h3>

Module output

</div>
</div>
</div>

</div>
```

As you can see the CSS options (-1, -2 and -3) are much leaner in code and make it easier to style the web pages. This author does not recommend using suffixes of 0 (default) or 1 unless absolutely needed.

To develop our template, we will put a module $style of "-2" on all of our modules:

```
<body>

  <div id="wrap">



 <div id="header">

  <?php echo $mosConfig_sitename; ?>

  <?php mospathway() ?>

  </div>



 <div id="main-body">

<div id="content">

  <div class="inside">

  <?php mosLoadModules('top',-2);?>

  <?php mosMainBody(); ?>

  </div></div>
```

```
  <div id="sidebar">
   <div class="inside">
   <?php mosLoadModules('left',-2);?>
   </div></div>


  </div> <!--end of main-body-->


<div id="sidebar-2">
   <div class="inside">
   <?php mosLoadModules('right',-2);?>
   </div></div>


  <div id="footer">
   <?php include_once( $mosConfig_absolute_path .'/includes/footer.php' ); ?>
   </div>


   </div> <!--end of wrap-->
</body>
```

Note that we cannot put these module styles on any of the following as they are not modules.

```
  <?php echo $mosConfig_sitename; ?>
  <?php mospathway() ?>
  <?php mosMainBody(); ?>
  <?php include_once( $mosConfig_absolute_path .'/includes/footer.php' ); ?>
```

Setting the modules to a CSS presentation has reduced the number of

tables to 14. Let's now add some simple styling to the template to get the result shown below:

First we will place the site title inside an <H1> tag. Its more semantically correct and will also help in SEO.

```
<h1><?php echo $mosConfig_sitename; ?></h1>
```

We will also add some CSS to style the modules with a border and a background for the header.

Our customize.css now looks like this:

```
/*Compass Design Customize.css file */

  * {

  margin:0;

  padding:0;

  }

  h1,h2,h3,h4,h5,h6,p,blockquote,form,label,ul,ol,dl,fieldset,address{

  margin:0.5em 0;

  }

  ul{

  margin-left:2em;

  }

  fieldset{

  padding:.5em;

  }

  body{

  font-size:76.1%;

  font-family:Verdana,Arial,Helvetica,sans-serif;

  line-height:1.3em;

  margin:1em 0;

  }

  #wrap{

  border:1px solid #999;

  background: url(../images/threecol-r.gif) repeat-y 75% 0;

  height:100% !Important;
height:1%;

  }

  #wrap-inner {
```

```
  background: url(../images/threecol-l.gif) repeat-y 25.125% 0;
  height:100% !Important;
height:1%;
  }
  #header{
  border-bottom: 1px solid #999;
  padding:10px;
  }
  #footer{
  border-top: 1px solid #999;
  padding:5px;
  }
  a{
  text-decoration:none;
  }
  a:hover{
  text-decoration:underline;
  }
  h1,.componentheading{
  font-size:1.7em;
  line-height:1.7em;
  }
  h2,.contentheading{
  font-size:1.5em;
  line-height:1.5em;
  }
  h3{
  font-size:1.3em;
  line-height:1.3em;
  }
  h4{
  font-size:1.2em;
  line-height:1.2em;
```

```
  }
  h5{
  font-size:1.1em;
  line-height:1.1em;
  }
  h6{
  font-size:1em;
  line-height:1em;
  font-weight:bold;
  }
  #footer,.small,.createdate,.modifydate,.mosimage_caption{
  font:0.8em Arial,Helvetica,sans-serif;
  color:#999;
  }
  .moduletable{
  margin-bottom:1em;
  padding:0 10px;
/*padding for inside text*/
  border:1px #CCC solid;
  }
  .moduletable h3{
  background:#666;
  color:#fff;
  padding:0.25em 0;
  text-align:center;
  font-size:1.1em;
  margin:0 -10px 0.5em -10px;
/*negative padding to pull h3 back out from .moduletable padding*/
  }
{mospagebreak title=Menus}
Menus
```

Setting that control how a menu is outputted are controlled in the module that publishes it. When you first make a module you will see a note that tells you that a module has been created with the same name.

In the module for that menu, there are several options for how the menu is presented:

- Vertical

The menu appears as a vertical table

- Horizontal

The menu appears as a horizontal table

- Flat List

The menu appears as a flat <ul> CSS list

The table format produces code such as:

```
<div class="moduletable">
<h3>Main Menu</h3>

<table width="100%" border="0" cellpadding="0" cellspacing="0">
<tr align="left">

<td>
<a xhref="index.php?option=com_frontpage&Itemid=1" class="mainlevel" id="active_menu">Home</a>


</td>

</tr>

<tr align="left">

<td>
<a xhref="index.php?option=com_content&task=view&id=5&Itemid=6"
class="mainlevel" >Joomla! License</a>
</td></tr> <tr align="left"><td><a xhref="index.php?option=com_content&task=section&id=1&Itemid=2"
class="mainlevel" >News</a>


</td>

</tr>
<tr align="left">
<td>
<a xhref="index.php?option=com_content&task=blogsection&id=0&Itemid=9"
class="mainlevel" >Blog</a>


</td>

</tr>

<tr align="left">

<td>
<a xhref="index.php?option=com_weblinks&Itemid=23" class="mainlevel" >Links</a>
</td></tr>
<tr align="left"><td>
<a xhref="index.php?option=com_contact&Itemid=3" class="mainlevel" >Contact Us</a>

</td>

</tr>
```

```
<tr align="left">

<td>
<a xhref="index.php?option=com_search&Itemid=5" class="mainlevel" >Search</a>

</td>

</tr>


<tr align="left">

<td>

<a xhref="index.php?option=com_newsfeeds&Itemid=7" class="mainlevel" >News Feeds</a>


</td>

</tr>

<tr align="left">
<td>
<a xhref="index.php?option=com_content&task=category&sectionid=3&id=7&Itemid=25"
class="mainlevel" >FAQs</a>

</td>

</tr>

<tr align="left"><td>
<a xhref="index.php?option=com_wrapper&Itemid=8" class="mainlevel" >Wrapper</a>

</td>

</tr>


</table>

</div>
```

The flat list will produce (for the same menu):

```
<div class="moduletable">


  <h3>Main Menu</h3>


  <ul id="mainlevel">


<li><a xhref="index.php?option=com_frontpage&Itemid=1"
class="mainlevel" id="active_menu">Home</a></li>

<li><a xhref="index.php?option=com_content&task=view&id=5&Itemid=6"
class="mainlevel" >Joomla! License</a></li>

<li><a xhref="index.php?option=com_content&task=section&id=1&Itemid=2"
class="mainlevel" >News</a></li>

<li><a xhref="index.php?option=com_content&task=blogsection&id=0&Itemid=9"
```

```
class="mainlevel" >Blog</a></li>

<li><a xhref="index.php?option=com_weblinks&Itemid=23" class="mainlevel" >Links</a></li>

<li><a xhref="index.php?option=com_contact&Itemid=3" class="mainlevel" >Contact Us</a></li>

<li><a xhref="index.php?option=com_search&Itemid=5" class="mainlevel" >Search</a></li>

<li><a xhref="index.php?option=com_newsfeeds&Itemid=7" class="mainlevel" >News Feeds</a></li>

<li><a xhref="index.php?option=com_content&task=category&
sectionid=3&id=7&Itemid=25" class="mainlevel"
>FAQs</a></li>

<li><a xhref="index.php?option=com_wrapper&Itemid=8" class="mainlevel" > Wrapper</a></li>

</ul>

</div>
```

Again, using CSS lists rather than tables results in reduced code and easier markup. After setting all our menus to flat lists we have only 12 tables (the rest cannot be removed withour hacking). Lists are also better than tables because text-based browsers, screen readers, non-CSS supporting browser, browsers with CSS turned off and search bots will be able to access your content more easily.

One of the other advantages of using CSS for menus is that there is alot of example code on various CSS developer sites. Lets look at one of them and see how it can be used.

A web page at maxdesign.com has a selection of over 30 menus all using the same underlying code. Its called the Listamatic. There is a slight difference in the code that we have to change in order to adapt these menus to Joomla.

These lists use the following code:

```
 <div id="navcontainer">


  <ul id="navlist">

 <li id="active"><a xhref="#" id="current">Item one</a></li>

 <li><a xhref="#">Item two</a></li>
 <li><a xhref="#">Item three</a></li>

 <li><a xhref="#">Item four</a></li>
 <li><a xhref="#">Item five</a></li>

 </ul>

 </div>
```

This means that there is an enclosing called navcontainer and the <ul> has an id of navlist.. To duplicate this effect in Joomla, we need to add a module location, let's use user1 before the left modules.

```
  <div id="sidebar">


<div class="inside">
```

```
<div id="navcontainer">

<?php mosLoadModules('user1',-2);?>

</div>

<?php mosLoadModules('left',-2);?>

</div></div>
```

Note that we place the mosLoadModules inside the
that matches those from ListaMatic. This could be any name, but for the
purposes of this tutorial, its useful to be able to easily use all
those menus!

Next, if there are any references to navlist, for Joomla this is output as mainlevel:
<ul id="mainlevel">

Lastly, we need to add a CSS module suffix to the user1 module in the admin backend so that module can receive unique CSS styles. Obviously this implies that we are putting the menu we want into this location.

A Module Class Suffix can be used on any module. When outputted, the names of the for that module will have the suffix appended.

So in this case:

- If it is using a -2 style it would be .moduletable-leftnav.
- If it is using a -3 style it would be .module-leftnav.

This use of a module class suffix is very useful. We will see in the
Tips and Tricks section that it allows different colored boxes with
just a simple change of the module class suffix.

For our site we will use List 10 by Mark Newhouse. Our CSS will be:

```
.moduletable-leftnav{

  margin-bottom:1em;

  padding:0; /*the padding is removed so the menu fills the whole module box*/

  border:1px #CCC solid;

  }

  .moduletable-leftnav h3{

  background:#666;

  width:100%;
```

```
  color:#fff;
  padding:0.25em 0;
  text-align:center;
  font-size:1.1em;
  margin:0;
/*now we have no padding in the module, we don't need the negative margins*/
  border-bottom: 1px solid #CCC;
  }
  #navcontainer{
  padding:0;
  color: #333;
  }
  #navcontainer ul{
  list-style: none;
  margin: 0;
  padding: 0;
  }
  #navcontainer li{
  border-bottom: 1px solid #ccc;
  margin: 0;
  }
  #navcontainer li a{
  display: block;
  padding: 3px 5px 3px 0.5em;
  border-left: 10px solid #333;
  border-right: 10px solid #9D9D9D;
  background-color:#666;
  color: #fff;
  text-decoration: none;
  }

  html>body #navcontainer li a {
   width: auto;
}
```

```
  #navcontainer li a:hover,a#active_menu:link,a#active_menu:visited{
  border-left: 10px solid #1c64d1;
  border-right: 10px solid #5ba3e0;
  background-color: #2586d7;
  color: #fff;
}
```

Designer's Tip

When trying to get a particular menu to work, here is a useful tip.
Create a default Joomla installation and then look at the code that is the mainmenu.
Copy and paste this code into an HTML editor (like Dreamweaver).
Replace all the links by "#" and then you can add CSS rules until the effect you want is acheived. The code for the menu to create the style is shown below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

  <html xmlns="http://www.w3.org/1999/xhtml">

  <head>

  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

  <title>Untitled Document</title>

  <style type="text/css">

  <!--
  .thisisyourCSS {
  /* put a rule here*/

  }
  -->

  </style>

  </head>

  <body>
```

```
<div class="moduletable">
<h3>Main Menu</h3>

<ul id="mainlevel">
<li><a xhref="#" class="mainlevel" id="active_menu">Home</a></li>
<li><a xhref="#" class="mainlevel" >Joomla! License</a></li>
<li><a xhref="#" class="mainlevel" >News</a></li>
<li><a xhref="#" class="mainlevel" >Blog</a></li>
<li><a xhref="#" class="mainlevel" >Links</a></li>
<li><a xhref="#" class="mainlevel" >Contact Us</a></li>
<li><a xhref="#" class="mainlevel" >Search</a></li>
<li><a xhref="#" class="mainlevel" >News Feeds</a></li>
<li><a xhref="#" class="mainlevel" >FAQs</a></li>
<li><a xhref="#" class="mainlevel" >Wrapper</a></li>

</ul>

</div>

</body>

</html>
```

Note the CSS is embedded rather than linked to make editing easier.
{mospagebreak title=Hiding Columns}
Hiding Columns

So far we have our layout such that we always have three columns, regardless of whether there is any content in there. We want to be able to "turn off" a column automatically, or "collapse" it if there is no content there.

The simplest way to do this is to have a small piece of PHP in the <head> of the page.

```
<?php if ( mosCountModules( 'right' ) <= 0) { ?>

<style type="text/css" media="screen">

#main-body {width:100%;}
#content{width:75%;}
#sidebar{width:25%;}
```

```
#sidebar-2{display:none;}

  </style>

 <?php } ?>
```

mosCountModules
will return the number of modules in that location. If it is equal or
less than zero, i.e., there is nothing there, then the style rules will
be adjusted. This php must appear AFTER the line that links to the template_css.css file. This is because if there are two identical CSS style rules, the one that is last will overwrite all the others.

This can also be done in a simliar fashion by having the if statement import a sub CSS file.
Hiding Module Code

When creating collapsible columns, it is good practice to set up the
modules to also not be outputted if there is no content there. If this
is not done the pages will have empty 's in them which can lead to cross browser issues.

To hide the empty , the following if statement is used:

```
<?php if (mosCountModules('left')) { ?><?php mosLoadModules( 'left', -2 );?>
 <?php } ?>
```

Anything can be placed inside the if statement, so we can put our there:

```
<?php if (mosCountModules('left')) { ?>

  <div id="sidebar">

    <div class="inside">

  <?php mosLoadModules( 'left', -2 );?>

</div></div>

 <?php } ?>
```

Using the above code, if there is nothing published in left, then #sidebar will not be outputted.

Conditional statements can also be used. Our sidebar column also had user1 in it as well as left, so we can have an OR statement:

```
<?php if (mosCountModules('left') || mosCountModules('user1')) { ?>

   <div id="sidebar">

   <div class="inside">

   <div id="navcontainer">

   <?php mosLoadModules('user1',-2);?>
```

```
</div>

<?php mosLoadModules('left',-2);?>

</div></div>

<?php } ?>
```

So if anything is published in either "left" or "user1" then this piece of code will be output.

Using this technique for our left and right columns, our index.php file now looks like this:

```
<?php defined( '_VALID_MOS' ) or die( 'Direct Access to this location is not allowed.' ); ?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="<?php echo _LANGUAGE; ?>"
xml:lang="<?php echo _LANGUAGE; ?>">

<head>
<meta http-equiv="Content-Type" content="text/html; <?php echo _ISO; ?>" />
<?php
if ($my->id) {
initEditor();
}
?>

<?php mosShowHead(); ?>

<script type="text/javascript"> </script>

<!--http://www.bluerobot.com/web/css/fouc.asp-->

<link href="templates/<?php echo $cur_template; ?>/css/template_css.css" rel="stylesheet"
type="text/css" media="screen" />

<?php if ( mosCountModules( 'right' ) <= 0) { ?>

<style type="text/css" media="screen">

#main-body {width:100%;}

#content{width:75%;}

#sidebar{width:25%;

#wrap{background:none;}
</style>

<?php } ?>
```

```
  </head>

  <body>

 <div id="wrap">

<div id="header">

  <h1><?php echo $mosConfig_sitename; ?></h1>
  <?php mospathway() ?>

  </div>

<div id="main-body">

<div id="content">

  <div class="inside">

  <?php mosLoadModules('top',-2);?>

  <?php mosMainBody(); ?>

  </div></div>

<?php if (mosCountModules('left') || mosCountModules('user1')) { ?>

  <div id="sidebar">

  <div class="inside">

  <div id="navcontainer">

  <?php mosLoadModules('user1',-2);?>

  </div>
  <?php mosLoadModules('left',-2);?>

  </div></div>

  <?php } ?>

</div> <!--end of main-body-->

<?php if (mosCountModules('right')) { ?>

  <div id="sidebar-2">

  <div class="inside">

  <?php mosLoadModules('right',-2);?>

  </div></div>
```

```
  <?php } ?>


<div id="footer">
<?php mosLoadModules(footer,-2);?>
</div>



  </div> <!--end of wrap-->

  </body>


</html>
```

I have also suggested a change to the footer at this point. The footer.php include makes it very difficult to change what this text is. I much prefer to have this placed as a module. There is even a module location called footer! Note in the template file that is associated with this guide, I have not changed this footer code in the downloadable template, but I show here giving the option should you wish to do so.
Designer's Tip

There are several names associated with modules in Joomla: banner, left, right, user1, footer etc. One important thing to realize is that the names do not corrospond to any particular location.
The location of a module is completely controlled by the template designer, as we have seen. Its customary to place them in a location that is connected to the name, but not required.
{mospagebreak title=Conclusion}
Conclusion

This tutorial goes through the steps of creating a joomla template. The template uses cascading style sheets (CSS) to produce a layout with minimal use of tables. This is a desirable goal as it means that the template code is easier to validate to World Wide Web Consortium (W3C) standards. It will also tend to load faster, be easier to maintain and perform better in search engines. The template also is source ordered for optimal performance in SEO.

{mospagebreak title=Appendix A:Tips and Tricks}
Appendix A: Tips and Tricks
Variable Page Widths, Rounded Corners, Text Resizers and More
Variable Page Widths

The design discussed here is a fluid page. It will resize with the browser window with a 10% page width margin on the left and right. Even though we have set the column widths to percentages, we can still have a fixed width page by setting #width
to an absolute number. Designing in this way means much more flexibility, its possible to change the page width by editing just one line of code.

One aspect to be careful of when working with fluid page design is images. Whichever direction you need to expand in, there always need to be a portion of solid color that can expand. Usually a small piece of

the image is sliced and then repeated.

As you can see in the example above, there is an area in the middle that can be scaled. The CSS might look like this:
background:url(../images/FluidImageSlicingMIDDLE.png) 0 0 repeat-x;

Compare this with a header that uses a photo:

There is no area here that can be repeated, and so this type of header would not work in a fluid design.
This principle applies to any sort of scaling, vertical
or horizontal. There must be a fraction of the image that can be
repeated.
Another advantage of having the template use percentage
based column widths is the use of a width resizer. By using seperate
stylesheets, its possible to dynamically change the page width based on
the user's preference. You can see this in action at Compass Design.
Rounded Corners

Probably the most common design look at the moment for modules is
using rounded corners for the boxes. The current standard is to use the
"-3" module style to produce 3 's. These three and then the main .module are used to place 4 images round the box.

You can see this in action at www.joomla.org. Here are the 4 images:

- http://www.joomla.org/templates/jw_joomla/images/jos_box_grey_bl.png
- http://www.joomla.org/templates/jw_joomla/images/jos_box_grey_br.png
- http://www.joomla.org/templates/jw_joomla/images/jos_box_grey_tl.png
- http://www.joomla.org/templates/jw_joomla/images/jos_box_grey_tr.png

And here is the code:

```
div.module-grey h3, div.moduletable-grey h3 {

font-family: Helvetica, Arial, sans-serif;

font-size: 1em;

font-weight: bold;

color: #333;

margin: -2px -8px 0 -8px;

border-bottom: 1px solid #cdcdcd;

padding-left: 10px;

padding-bottom: 2px;

}

div.module-grey, div.moduletable-grey {
```

```
background: url(../images/jos_box_grey_tl.png) 0 0 no-repeat;
margin: 0;
padding: 0;
margin-bottom: 20px;
}
div.module-grey div, div.moduletable-grey div {
background: url(../images/jos_box_grey_tr.png) 100% 0 no-repeat;
}
div.module-grey div div, div.moduletable-grey div div {
background: url(../images/jos_box_grey_bl.png) 0 100% no-repeat;
}
div.module-grey div div div, div.moduletable-grey div div div {
background: url(../images/jos_box_grey_br.png) 100% 100% no-repeat;
padding: 8px;
width: auto !important;
width: 100%;
}
div.module-grey ul, div.moduletable-grey ul {
margin: 10px 0;
padding-left: 20px;
}
```

Many techniques exist to produce rounded corners. A couple of sites that have summaries of them are css-discuss.incutio.com and Smiley Cat.
Fixed Width Boxes with Two Images

First developed at Joomlashack,
its possible to half the number of images used in a box. This technique
will give boxes that scale vertically but not horizontally.

The two images used are:
and
The CSS used to produce the effect is:

```
.moduletable-box {
  margin:0 0 10px 0;
  padding:0 0 10px 0;
  background:url(../images/bottom.png) bottom left no-repeat;
```

```
 }

 .moduletable-box h3 {

 padding:8px 10px 6px 15px;

 margin-bottom:8px;

 text-align:left;

 font:bold 1.1em Arial,Helvetica,sans-serif;

 color:#fff;

 background:url(../images/top.png) top left no-repeat;

}
```

This technique is particulary useful for a number of reasons:

- The bottom box image can be reused with different colors
- The H3 heading's background will scale with text resizing (joomla.org technique does not)
- It is very lean in code
- Half as many images are twice as fast (roughly) to load

It requires the module $style suffix to be "-2" to work correctly.

Fluid Width and Height Boxes with Two Images

A more advance technique has been developed by Compass Design. It requires the module $style suffix to be "-3" to work correctly. Again, each is used to place the corner of a box, but only two images are used. A module class suffix of -box has been used here.

and

```
.module-box {

 background: url(../images/boxright.png) top right no-repeat;

 padding:0;

 margin:0 0 10px 0;

 }

 .module-box h3 {

 margin:0;

 padding:0 0 4px 0;

 border-bottom:#ccc 1px solid;

 color: #666;

 font: bold 1em Tahoma, Arial, Helvetica, sans-serif;

 text-align:center;

 }

 .module-box div {
```

```
 background: url(../images/boxleft.png) top left no-repeat;

 margin:0;

 padding:6px 0 0 0;

 }

 .module-box div div{

 background: url(../images/boxleft.png) bottom left no-repeat;

 padding:0 0 0 5px;

 }

 .module-box div div div{


 background: url(../images/boxright.png) bottom right no-repeat;

 padding:0 5px 5px 0;

 height:auto !important;
height:1%;

}
```

This technique will scale vertically and horizontally.

Almost any technique from the two sites given earlier should work. Of paricular interest is 456bereastreet's Flexible box with custom corners and borders. This only uses one image so it would be easy to change colors. The implementation of this technique is left as an excercise for the reader!

Text Resizers

Many size are becoming more accessible and are incorperating text resizers into the design. As a general rule for best cross-browser consistancy, you need to do a few of things for text resizer buttons to work well:

1. in the body tag, define a font size of 76%

```
body {
font-size:76%;
}
```

2. Define all font sizes in "em". This is a relative unit, for example:

```
p {
font-size:1em;
}
```

3. Make all text containers dynamic. An example of this not working is www.joomla.org. If you make bigger, the container for the horizontal menu does not get bigger, if you use FF to go past the size that you can acheive with the buttons (ctrl+ in FF), the layout breaks. You get round it by not putting any fixed heights on containers, you can see this working at www.joomlashack.com.

4. Download joomla.org's font style switcher file

(http://forum.joomla.org/Themes/joomla/md_stylechanger.js)

5. Put that file somewhere in the folder of the template you are using

6. Put A+, A-, and Reset images in your template's image folder

7. Paste the following code snippet somewhere in your template's index.php file

```
<script type="text/javascript" language="javascript" xsrc="<?php echo $mosConfig_live_site;?>
/templates/<?php echo $mainframe->getTemplate(); ?>/____1____">

</script>

<a xhref="index.php" title="Increase size" onclick="changeFontSize(1);return false;">
<img xsrc="<?php echo $mosConfig_live_site;?>/templates/<?php echo $mainframe->getTemplate(); ?>
/images/____2____" alt="" border="0" />
</a>
<a xhref="index.php" title="Decrease size" onclick="changeFontSize(-1);return false;">
<img xsrc="<?php echo $mosConfig_live_site;?>/templates/<?php echo $mainframe->getTemplate(); ?>
/images/____3____" alt="" border="0" />
</a>
<a xhref="index.php" title="Revert styles to default" onclick="revertStyles(); return false;">
<img xsrc="<?php echo $mosConfig_live_site;?>/templates/<?php echo $mainframe->getTemplate(); ?>
/images/____4____" alt="" border="0" />
</a>
```

8. Do all of the following:

- Replace ____1____ with the location in your template folder where you saved the .js file
- Replace ____2____ with the name of your A+ image
- Replace ____3____ with the name of your A- image
- Replace ____4____ with the name of your Reset image

Thanks to r0tt3n and his FAQ at the Joomla forums for portions of this technique.

Drop Down Menus

For various reasons, SEO and accessibility amoung them, the author is biased against flash and JavaScript. Better to emphasize W3C valid code and lean pages, neither of which is helped by these two approaches. How then to acheive menus that duplicate these effects, like a drop down menu.

Well, there are a number of techniques you can use with CSS to get more visually attractive menus, all of them use unordered lists (bulleted lists) to create the menu. Let’s look at one, a drop down menu.

The menu is what has been coined "suckerfish", its pure CSS, very lean, hack free and just has 12 lines of JavaScript for IE.

You can see a demo here:

www.htmldog.com/articles/suckerfish/dropdowns/example/

You can find guides to how the thing works at a couple of sites:

www.htmldog.com/articles/suckerfish/dropdowns/

www.alistapart.com/articles/dropdowns/

Now, you might have noticed that you need your menu outputted as a good clean list. Well it just so happens that there is a module out there to do this, and we'll need it. Its called extended_menu, you can find it here:

de.siteof.de/extended-menu.html

It's easiest if you give it a menu and module class suffix. I used "mainnav" for both (you'll see in the CSS below). A couple of other settings you will need:

- Menu style: Tree List
- Expand Menu: Yes

```
.moduletablemainnav{
    position:relative;
    z-index:100;
    font:0.9em Verdana, Arial, Helvetica, sans-serif;
    margin:0;
    padding:0;
    }
    #mainlevelmainnav,#mainlevelmainnav ul{
    float:left;
    list-style:none;
    line-height:1em;
    background:transparent;
    font-weight:700;
    margin:0;
    padding:0;
    }
```

```
#mainlevelmainnav a{
display:block;
color:#f90;
text-decoration:none;
margin-right:15px;
padding:0.3em;
}
#mainlevelmainnav li{
float:left;
padding:0;
}
#mainlevelmainnav li ul{
position:absolute;
left:-999em;
height:auto;
width:11em;
font-weight:400;
background:#36f;
border:#00C 1px solid;
margin:0;
}
#mainlevelmainnav li li{
width:11em;
}
#mainlevelmainnav li ul a{
width:11em;
color:#fff;
font-size:0.9em;
line-height:1em;
font-weight:400;
}
#mainlevelmainnav li:hover ul ul,#mainlevelmainnav li:hover ul ul ul,
#mainlevelmainnav li.sfhover ul ul,#mainlevelmainnav li.sfhover ul ul ul{
```

```
    left:-999em;

    }

    #mainlevelmainnav li:hover ul,#mainlevelmainnav li li:hover ul,
    #mainlevelmainnav li li li:hover ul,#mainlevelmainnav li.sfhover ul,
    #mainlevelmainnav li li.sfhover ul,#mainlevelmainnav li li li.sfhover ul{

    left:auto;

    z-index:6000;

    }

    #mainlevelmainnav li li:hover,#mainlevelmainnav li li.sfhover{

    background:#039 url(../images/soccerball.gif) 98% 50% no-repeat;

  }
```

Now,
just make sure you have the z-indexes set up properly, also remember to have a z-index, the element needs some sort of positioning, if not absolute then relative.

Last but not least you need to add the JavaScript for IE into the head of the template index.php (or a js file), it does not interpret :hover correctly.

```
  <script type="text/javascript"><!--//--><![CDATA[//><!--
  sfHover = function() {
 var sfEls = document.getElementById("mainlevelmainnav").getElementsByTagName("LI");
 for (var i=0; i<sfEls.length; i++) {
 sfEls[i].onmouseover=function() {
 this.className+=" sfhover";
 }
 sfEls[i].onmouseout=function() {
 this.className=this.className.replace(new RegExp(" sfhover\\b"), "");
 }
 }
 }
 if (window.attachEvent) window.attachEvent("onload", sfHover);
 //--><!]]></script>
```

This technique is discussed further at the Joomla forums.