

Theme guide (Drupal 6)

Welcome to the theming handbook. The following pages will cover the core ideas behind theming as it applies to Drupal 6 and beyond. Theming instructions for [versions 5 and below](#) are still available. For an overview of the changes see the [upgrade guide](#).

This handbook will illustrate some of the differences from previous versions where appropriate. It is presented from a themers perspective. If you are developing a module that outputs presentation data, then please read the [theme section](#) in the module developers guide. All output should be themable.

Note to documentation contributors. Please don't add HOWTO sections and instructions related to contributed modules or themes. Keep the information here related to core and only link to their pages instead when it makes sense to do so.

Introduction to theming

Drupal is known for its power and flexibility and it can be a daunting experience trying to understand how it all comes together. There are countless ways to solve theming problems but not all are recommended. Knowing the "Drupal way" can lead to minimized code bloat and easier maintenance. If you choose to break the rules and go your own path, knowing them first may lead to a greater chance of success.

This does not mean you must understand Drupal top to bottom in order to create your theme. You can dive in as deep as you need to get the job done, but the more complex your site's design, the better it is to understand more of what Drupal's theming system provides.

The aim of this handbook is to expose all parts of Drupal's theming layer. Some areas will be geared towards more technical users while others will be more basic. The information presented in the following pages will progress from a general overview all theme developers should be familiar with to the more specific, and at times, more technical explanations.

A few things you should be aware of before proceeding:

- An understanding of [xHTML](#) and [CSS](#).
- JavaScript and jQuery if your theme needs scripting support.
- The [terminology](#) used in Drupal.

Knowing [PHP](#) will be a requirement in some situations but it is possible to avoid it entirely with pure CSS based themes.

Depending on your goals for the theme, it can be a very simple or complex process. Drupal is very open-ended so carefully consider what you are trying to achieve. You should think about the requirements of the site first. It is a lot easier to design with specific requirements in mind than creating a general purpose theme.

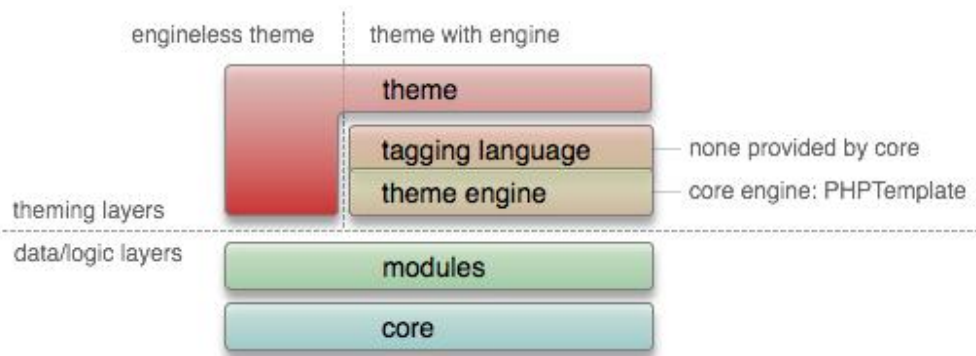
If you ever get stuck on a problem, read the [troubleshooting page](#), ask in the [theming forum](#) or on IRC @ #drupal-themes. See [How to effectively use IRC](#) for instructions.

Theming overview

A common software practice is to compartmentalize the programming layer from the presentation layer. There are [numerous reasons](#) for this, the most obvious being that the skill set required for programming the back-end [business logic](#) is very different from creating a visually appealing and effective user interface. As a theme developer, you can control certain aspects of the available data, but it is limited to the output and presentation. Only Drupal core and modules should work with input. For example, a module can implement a form with a default look and feel and handle user input, saving it to the database. The theme's role is only to override the look and feel.

This abstraction in Drupal is achieved through the [theme](#) function. It is a conduit to the theming subsystem. It allows theme engines to provide an optional middle layer for tagging languages such as [PHPTAL](#) or [Smarty](#). It also allows themes to control all presentation markup. Theme engines are optional as are tagging languages. PHPTemplate is the default engine. As the name suggests, it uses the language PHP when outputting variables mixed in with the [xHTML](#) markup.

Starting with Drupal 6, the requirements for creating theme engines have been lowered substantially.



All layers can implement a themed representation of the output, but (with a few exceptions) only within the theming layers can overrides occur. Theme engines can override themed output from core and modules, while themes can override everything else.

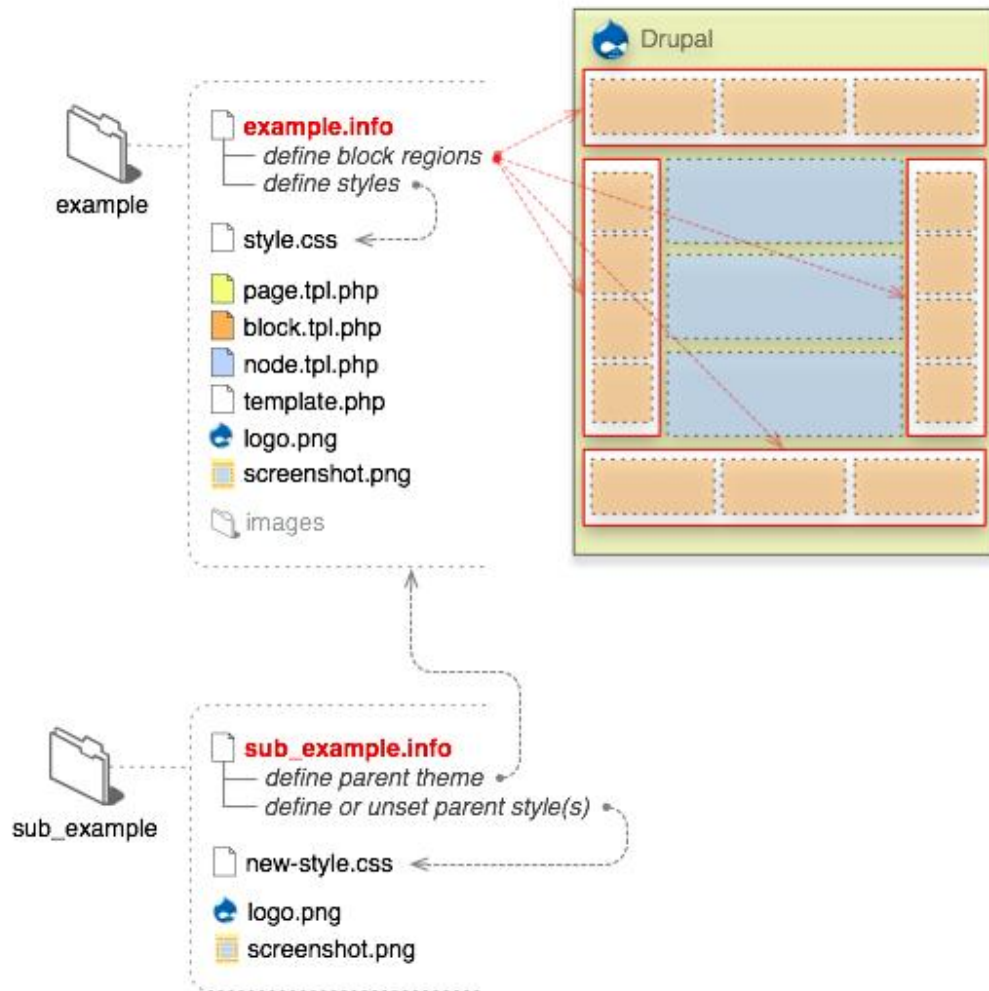
Note that the PHPTemplate engine does not override any output but other theme engines can. There is a special case where a module can influence the output or override absolutely everything, but it is for very special cases and should not interfere in most situations. For example, the [devel](#) themer module does this to [aid theme development](#). More details will be discussed in a [later section](#).

You can ignore most of this if your theme will be [styled entirely through CSS](#), but when the markup needs to be altered it is important to know how to [get to the source of the output so it can be altered](#).

- Note that the Drupal core and countless modules always use [themable functions](#) and [template files](#) to output the presentation markup. Never hack on files outside your theme folder as that would lead to complications when there is a need to update. Doing this is known as "forking". The power of open source lies in having the community manage bug fixes and add new features. Once you start a fork you create a closed system, and you lose the leverage of the community. Drupal provides all the functionality to override the presentation layer. If you ever have to hack on files outside the theme, either you are doing something wrong or you have found a bug. In the latter case, please file a [bug report](#). Or even better, [provide a patch file](#) to fix the problem.
- For those familiar with the PHPTemplate engine in previous incarnations, almost all the functionality has been moved deeper into core. The job of PHPTemplate now is to only discover theme functions and templates on behalf of the theme. It is less of an engine and more of a theme helper. PHPTemplate was originally written by [Adrian Rossouw](#) for 4.7. The changes in 6 were the work of [Earl Miles](#). [An extended forum discussion](#) provides some of the reasoning behind the initial creation of the engine and the [issue queue for the new direction](#) in 6.

Anatomy of a Drupal theme

phptemplate theme:



.info (required)

All that is required for Drupal to see your theme is a ".info" file. Meta data, [style sheets](#), [JavaScripts](#), [block regions](#) and more can be defined here. Everything else is optional.

The internal name of the theme is also derived from this file. For example, if it is named "drop.info", then Drupal will see the name of the theme as "drop". *Drupal 5 and below used the name of the enclosing folder of the theme.*

Info files for themes are new in Drupal 6. In version 5, .info files were used solely for modules.

template files (.tpl.php)

These templates are used for the xHTML markup and PHP variables. In some situations they may output other types of data --[xml rss](#) for example. Each .tpl.php file handles the output of a specific themeable chunk of data, and in some situations it can handle multiple .tpl.php files through [suggestions](#). They are optional, and if none exists in your theme it will fall back to the default output. Refrain from having complex logic in these files. In most cases, it should be straight xHTML tags and PHP variables. A handful of these templates exist in directories where core and contributed modules exist. Copying them to your theme folder will force Drupal to read your version.

Note: The [theme registry](#) caches information about the available theming data. You must reset it when adding or removing template files or theme functions from your theme.

template.php

For all the conditional logic and data processing of the output, there is the template.php file. It is not required, but to keep the .tpl.php files tidy it can be used to hold [preprocessors](#) for generating variables before they are merged with the markup inside .tpl.php files. Custom functions, [overriding theme functions](#) or any other customization of the raw output should also be done here. This file must start with a PHP opening tag "<?php", but the close tag is not needed and it is recommended that you omit it.

Sub-themes

On the surface, sub-themes behave just like any other theme. The only difference is that they inherit the resources from their parent themes. To create one, a "base theme" entry inside the .info file is needed. From there it will inherit

the resources from its parent theme. There can be multiple levels of inheritance; i.e., a sub-theme can declare another sub-theme as its base. There are no hard set limits to this.

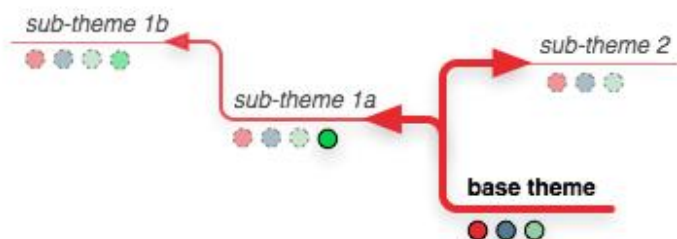
Drupal 5 and below required sub-themes to be in sub-directories of the parent theme. This is no longer the case.

Others

- The logo and screen shot is not absolutely necessary for the theme to function, but it is recommended, especially if you are [contributing your theme to the Drupal repository](#). Screenshots will show inside the theme administration page and the user account settings for selecting themes when the appropriate permissions are set. See the [screenshot guidelines](#) for more information.
- To supply administrative UI settings or "features" beyond logo, search, mission, etc., a "theme-settings.php" file can be used. This is an advanced feature. More information can be found in the [Advanced settings](#) handbook page.
- For [color module support](#), a "color" directory with a "color.inc" file is needed along with various support files.
- *If you want to base your work on a core theme, use [sub-theming](#) or make a copy and rename the theme. Directly modifying Garland or Minnelli is strongly discouraged, since they are used for the install and upgrade process.*
- *All themes should be installed in the "sites/all/themes" directory to separate them from core files. Read about [multi-site installations](#) to see all the possible installation directories.*

Sub-themes, their structure and inheritance

Sub-themes are just like any other theme, with one difference: They inherit the parent theme's resources. There are no limits on the chaining capabilities connecting sub-themes to their parents. A sub-theme can be a child of another sub-theme, and it can be branched and organized however you see fit. This is what gives sub-themes great potential.



Imagine starting with a base theme designed as wireframes, then applying and refining all the details from a sub-theme. Then, off the same wireframe, testing out alternate designs by branching out another sub-theme. Working on a multi-site installation but you need a cohesive look and feel? With sub-theming, a lot of the design resources can be shared. Site-specific changes can be set to a specific sub-theme, but any shared resources can be edited once to be applied across all the installations. With careful planning, the possibilities are endless.

To declare a parent or "base theme", set this inside the sub-theme's [.info file](#), "themeName" being the internal name of the parent theme:

```
base theme = themeName
```

The following are inherited:

- All [style sheets](#) defined in the parent theme, but there is an [option](#) so this can be controlled.
- All [JavaScripts](#) defined in the parent theme.
- All template (*.tpl.php*) files.
- Anything defined from the "template.php" file. This includes [theme function overrides](#), [preprocessors](#) or anything else. Each sub-theme will include its own template.php file along with the one provided by its parent.
- Screen shot for the parent theme as long as the .info file uses the same settings from its parent.

The following are not inherited:

- logo.png set for the theme. This does not include uploaded logos, since they will always be used.
- Some of the settings set within the .info file. This includes regions. If you are not using the default regions, then make sure your "page.tpl.php" file for the sub-theme matches what is set for regions in the .info file. Each base and sub-theme can have their own region settings.
- Anything set within the "theme-settings.php" file for the [advanced theme settings](#).
- Anything set within the "color" directory for [color.module support](#).

Note that sub-themes can be placed outside of the base theme's directory or within it. Before version 6, they had to be in the sub-directory of the parent theme.

.info files for themes

New to Drupal 6, an .info configuration file is now required for each theme. This file should reside in your theme's directory. Without this file, your theme will not be visible to Drupal. The .info file must end with the ".info" extension.

The internal "machine" readable name of the theme is derived from this file. For example, if the file is named "drop.info", then Drupal will see the name of the theme as "drop". Be sure that no odd characters are used, since it tells Drupal how to form various [functions in PHP](#) and therefore it will have the same limitations. The name should start with an alphabetic character, no spaces or punctuation. Underscores are allowed but no hyphens. Numbers are allowed as long as a number is not the first character.

Notes:

- *Warning! Modules bearing the same internal name as your theme can prevent your site from functioning. This may happen from duplicate function names, which are illegal in PHP. All installed components must have unique names.*
- *The contents of the .info file is cached in the database, so altering it will not be noticed by Drupal. (Do not confuse the cache with the [theme registry](#).) To clear the cache, do one of the following:*
 1. *Click the "clear" button located at "Administer > Site configuration > Performance".*
 2. *With devel block enabled (comes with devel module), click the "Empty cache" link.*
 3. *Simply visit the theme select page at "Administer > Site building > Themes".*

The syntax is similar to [INI](#) files. The .info file is basically a static text file for configuring the theme. Each line in the text file is a key-value pair with the key on the left and the value on the right, with an "equals sign" between them. (*example: key = value*) Semicolons are used to comment out a line. Some keys use a special syntax with square brackets for building a list of associated values, referred to as an "array". If you are unfamiliar with arrays, using them should be apparent enough by following the example of the default .info files that come with Drupal and reading the explanations of the examples that follow.

Drupal understands the keys listed below. Drupal will use [default values](#) for the optional keys not present in the .info file. See the [examples](#) set for core themes.

- [name](#) !
- [description](#) *
- [screenshot](#)
- [version](#) *
- [core](#) !
- [engine](#) *
- [base theme](#)
- [regions](#)
- [features](#)
- [stylesheets](#)
- [scripts](#)
- [php](#)

name (*required*)

The human readable name can now be set independently from the internal "machine" readable name. This imposes fewer restrictions on the allowed characters.

name = Un tema nombre de fantasía

description (*recommended*)

A short description of the theme. This description is displayed on the theme select page at "Administer > Site building > themes".

description = Tableless multi-column theme designed for blogs.

screenshot

The optional screenshot key tells Drupal where to find the theme's thumbnail image, used on the theme selection page (admin/build/themes). If this key is omitted from the .info file, Drupal uses the "screenshot.png" file in the theme's directory.

Use this key only if your thumbnail file is not called "screenshot.png" or if you want to place it in a directory outside of your theme's base directory (e.g. screenshot = images/screenshot.png).

screenshot = screenshot.png

version (*recommended*)

The version string will automatically be added by drupal.org when a release is created and a tarball packaged. So you may omit this value for contributed themes. However, if your theme is not being hosted on the drupal.org infrastructure, you can give your theme whatever version string makes sense.

version = 1.0

core (required)

From 6.x onward, all .info files for *modules and themes* **must indicate** what major version of Drupal core they are compatible with. The value set here is compared with the [DRUPAL_CORE_COMPATIBILITY](#) constant. *If it does not match, the theme will be disabled.*

core = 6.x

The drupal.org packaging script automatically sets this value based on the Drupal core compatibility setting on each release node. So people downloading packaged themes from drupal.org will always get the right thing. However, for sites that deploy Drupal directly from CVS, it helps if you commit this change to the .info file for your theme. This is also a good way to indicate to users of each theme what version of core the [HEAD](#) of CVS is compatible with at any given time.

engine (recommended)

The theme engine, which is used by the theme. If none is provided, the theme is assumed to be stand alone, i.e., implemented with a ".theme" file. Most themes should use "phptemplate" as the default engine.

PHPTemplate's job is to discover theme functions and templates for the behavior of the theme. Omit this entry only if you know what you are doing.

engine = phptemplate

base theme

Sub-themes can declare a base theme. This allows for theme inheritance, meaning the resources from the "base theme" will cascade and be reused inside the sub-theme. Sub-themes can declare other sub-themes as their base, allowing multiple levels of inheritance. Use the internal "machine" readable name of the base theme. The following is used in Minnelli, the sub-theme of Garland.

base theme = garland

More details are available on the page [Sub-themes, their structure and inheritance](#).

regions

The block regions available to the theme are defined by specifying the key of 'regions' followed by the internal "machine" readable name *in square brackets* and the human readable name as the value, e.g., regions[theRegion] = The region name.

If no regions are defined, the following values are assumed. You can override the values for your specific needs.

regions[left] = Left sidebar
regions[right] = Right sidebar
regions[content] = Content
regions[header] = Header
regions[footer] = Footer

More details are available on the page [Blocks, content and their regions](#).

features

Various page elements output by the theme can be toggled on and off on the theme's configuration page. The "features" keys control which of these check boxes display on the theme's configuration page. This is useful for suppressing check boxes for elements not defined or used by a theme. To suppress a check box, omit the entry for it. However, if none are defined, all the check boxes will display due to the assumed defaults.

The example below lists all the available elements controlled by the features key. By commenting out the primary_links and secondary_links elements, their check boxes are suppressed and are not seen by site administrators.

features[] = logo
features[] = name
features[] = slogan
features[] = mission
features[] = node_user_picture
features[] = comment_user_picture
features[] = search

```

features[] = favicon
; These last two disabled by redefining the
; above defaults with only the needed features.
; features[] = primary_links
; features[] = secondary_links

```

More details are available on the page [Custom theme settings](#).

stylesheets

Traditionally, themes default to using style.css automatically and could add additional stylesheets by calling [drupal_add_css\(\)](#) in their template.php file. Starting in 6, themes can also add style sheets through their .info file.

```
stylesheets[all][] = theStyle.css
```

More details are available in the [style sheets](#) section.

scripts

Traditionally, themes could add javascripts by calling [drupal_add_js\(\)](#) in their template.php file. Starting in 6.x, themes can also add javascripts by adding lines to their .info file:

```
scripts[] = script.js
```

More details are available in the [JavaScript & jQuery](#) section.

php

This defines the minimum **PHP version** the theme will support. The default value is derived from the [DRUPAL_MINIMUM_PHP](#) constant, which is the minimum required version for the rest of core. This can be redefined for a newer version if needed. For most themes, this should not be added.

```
php = 4.3.3
```

Example .info files from core themes

Garland:



```

; $Id: garland.info,v 1.5 2007/07/01 23:27:32 goba Exp $
name = Garland
description = Tableless, recolorable, multi-column, fluid width theme (default).
version = VERSION
core = 6.x
engine = phptemplate
stylesheets[all][] = style.css
stylesheets[print][] = print.css

```

```

; Information added by drupal.org packaging script on 2008-02-13
version = "6.0"
project = "drupal"
datestamp = "1202913006"

```

Minnelli *sub-theme of Garland*:

```
; $Id: minnelli.info,v 1.7 2007/12/04 20:58:44 goba Exp $
```

```
name = Minnelli
description = Tableless, recolorable, multi-column, fixed width theme.
version = VERSION
core = 6.x
base theme = garland
stylesheets[all][] = minnelli.css
```

```
; Information added by drupal.org packaging script on 2008-02-13
version = "6.0"
project = "drupal"
datestamp = "1202913006"
```

Note that everything from the line "; Information added by drupal.org packaging script on 2008-02-13" and down is added by the drupal.org packaging script. You should never manually add the project and datestamp keys. The version key added manually (in the first section) allows sites to use your theme when taken directly from CVS.

Default .info values

The following are the assumed defaults. When they are not defined, the theme will automatically take these values.

Note: These defaults apply as a group. In other words, overriding a region with regions[sub_header] = Sub-header will omit the rest of the default regions. To gain them back, they must be redefined. Even with "stylesheets". Even though it's not technically in a group, defining another stylesheet will prevent "style.css" from being included unless it is redefined.

regions

```
regions[left] = Left sidebar
regions[right] = Right sidebar
regions[content] = Content
regions[header] = Header
regions[footer] = Footer
```

features

```
features[] = logo
features[] = name
features[] = slogan
features[] = mission
features[] = node_user_picture
features[] = comment_user_picture
features[] = search
features[] = favicon
features[] = primary_links
features[] = secondary_links
```

stylesheets

```
stylesheets[all][] = style.css
```

scripts

```
scripts[] = script.js
```

screenshot

```
screenshot = screenshot.png
```

php (minimum support)

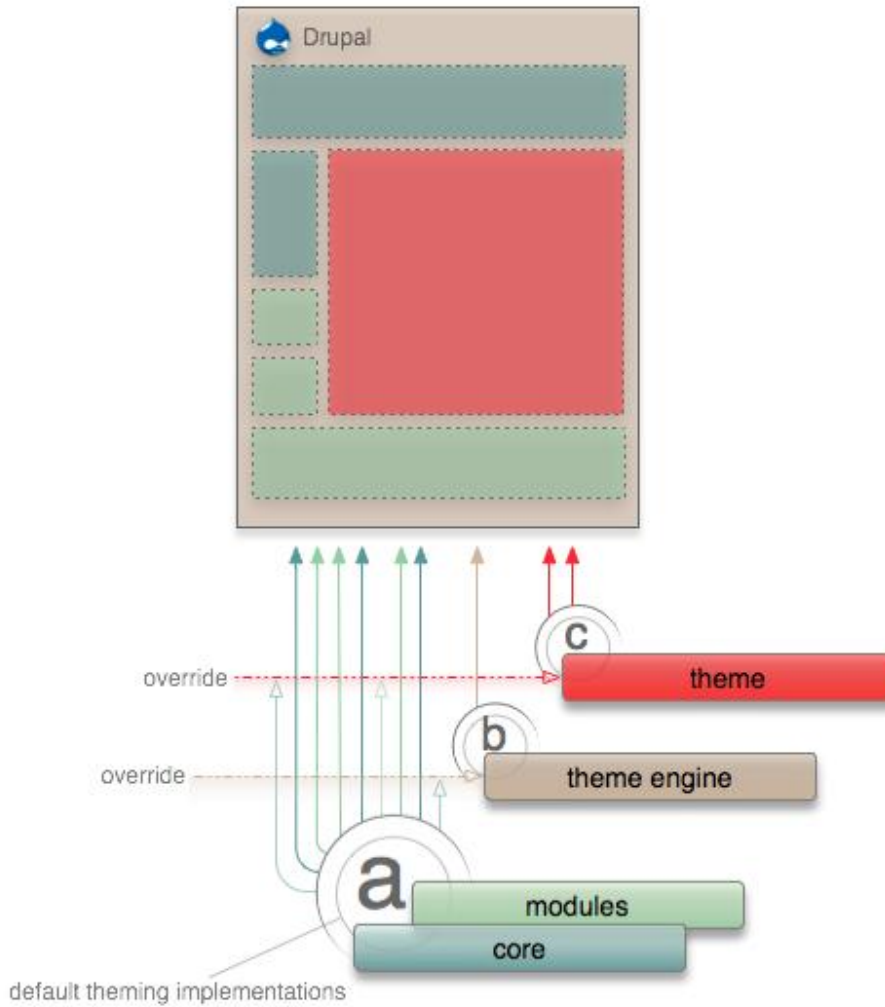
[DRUPAL_MINIMUM_PHP](#) is a constant. It points the the minimum requirements for Drupal core to run.

```
php = DRUPAL_MINIMUM_PHP
```

Overriding themable output

The following only applies when the default markup needs changes. This section can be skipped if the presentation is handled only through [style sheets](#).

There are three aspects to overriding the themed output. The first is knowing where the [source originates](#), the second is

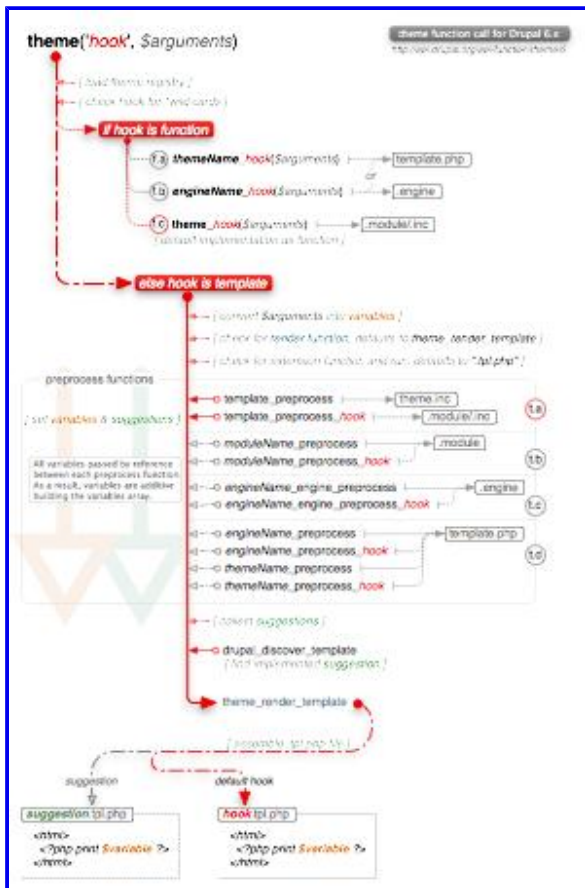


If you need more control beyond this convention of overrides, the [theme registry can be manipulated](#) for more control.

Note: Although it is still possible, PHPTemplate.engine in Drupal 6 no longer overrides theme functions. In 5, that is what allowed templates to be used for a handful of the theming hooks. It is no longer necessary.

3. Functions vs. templates:

There are two ways or "types" as it was mentioned above in implementing any given hook. Normal "*functions*" or "*templates*". Depending on the nature of the themed element, one may be better suited than the other. Core and modules can construct the output in either type. The upper theming layers can carry over the same type or change it.



Links to PDF. [Flow map for 5](#) also available for comparison.

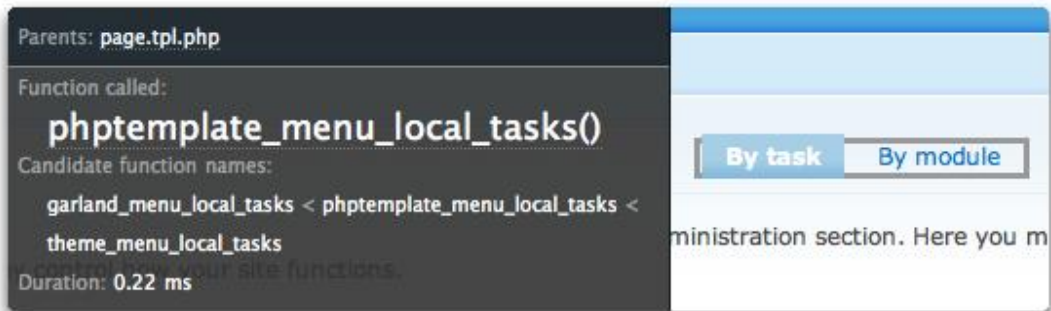
Theming hooks implemented as functions provide speed. It is about five times faster than templates but they are difficult for designers who may be more familiar with working directly in xHTML. Generally, this should not be a concern as it depends on the nature of the hook and the number of times it is called on a page.

Note: In versions before 6, core and modules could only work as functions for the theming hooks. Implementing through templates was done by PHPTemplate by overriding the hooks and doing the conversion on the engine layer.

Here are two examples on overriding with the help of devel themer.

Overriding functions:

The theme function [theme_menu_local_tasks](#) is a simple function for outputting both primary and secondary tabs. The theming hook in this case is "menu_local_tasks". To override, the "theme" prefix in the function name can be changed to the name of your theme or the theme engine the theme is running under. Using the theme name is recommended due to potential name collisions with [sub-themes](#).



The example shows Garland is using the name of the theme engine for the override. If you were to create a sub-theme based on Garland, then using the name of your theme is mandatory.

Placing the following inside the theme's template.php file will override the default after clearing the theme registry. Change "drop" to the name of your theme.

```

<?php
function drop_menu_local_tasks() {
  $output = "";

  if ($primary = menu_primary_local_tasks()) {
    $output .= "<ol class='tabs primary'>\n". $primary . "</ol>\n";
  }
  if ($secondary = menu_secondary_local_tasks()) {
    $output .= "<ol class='tabs secondary'>\n". $secondary . "</ol>\n";
  }

  return $output;
}
?>

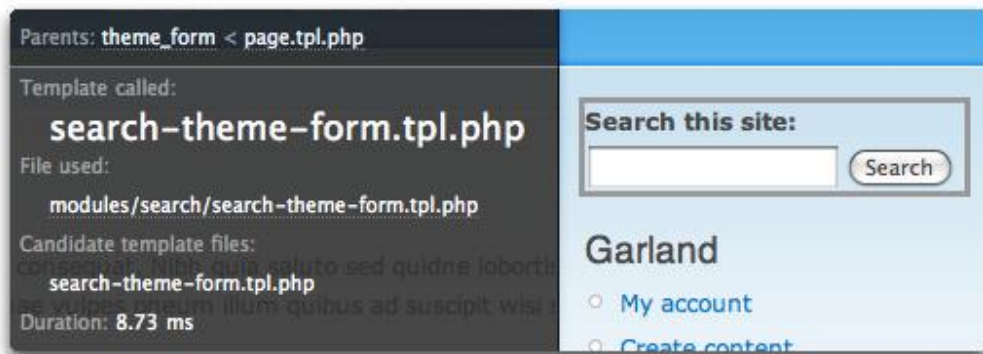
```

The only change here is the markup from an unordered list to an ordered list.

A listing of all the [theme functions](#) can be found on [api.drupal.org](#).

Overriding templates:

If the default implementation is done as a template then simply moving the source template file into the theme will automatically override it after clearing the theme registry. Here is an example for [search-theme-form.tpl.php](#). Note that the theming hook in this case is "search_theme_form" with the template using hyphens instead of underscores.



That is all you need to do. Open the copied template in your editor to make your alterations. All the core .tpl.php files are documented. It should give a good indication on what can be done with the output.

Note: templates can be placed in any directory within the theme. This allows for better management and less clutter in the base level of the theme directory.

Related pages:

- To customize the variables inside templates, see the sub-page on [Preprocess functions](#).
- A listing of all the theme templates can be found on the sub-page, [Core templates and suggestions](#).

Converting from functions to templates

Converting a theme function into a template takes some initial work but once it is done, it's easier to work with. If you are collaborating with a designer, the conversion will enable them to focus on designing, not coding. There are many templates already available in core and many more will be converted in future versions. Contributed modules that follow best practices should also have templates available. These instructions are provided for the theming hooks not already made available as templates.

Getting Drupal to recognize the theming hook as a template is automated. These are the only requirements to trigger this change:

- Name of the template must match the theming hook.
- The underscores in the hook must be changed to hyphens.
- The template name must have an extension of ".tpl.php". (*It can vary based on the theme engine.*)

Consider the theming function [theme_user_signature](#). The theme hook here is "user_signature". Creating a file named "user-signature.tpl.php" will tell Drupal that the hook is now a template after clearing the registry. Any content in this file will now take the place of the function. The part that takes more work is setting up the variables to be used in this file and that is done through preprocess functions.

A few notes:

- While it is possible to code directly inside the template, it is not considered good practice. All the complex logic should be separated from the `.tpl.php` file. This keeps it clean and easy to manage.
- There is also the issue of security. The separation can minimize the chance of cross-site scripting attacks by cleaning out potentially malicious user generated content. When handing over template files to your designer, all the output should be clean so they do not have to concern themselves with security issues.
- Compare the [theming functions in 5](#) to the [template conversions in 6](#) for forums. You can use that as an example on converting between the two types.
- More information on [preprocess functions](#) available.

The theme registry:

Drupal's theme registry maintains cached data on the available theming hooks and how to handle them.

For most theme developers, the registry does not have to be dealt with directly. Just remember to clear it when adding or removing theme functions and templates. Editing existing functions and templates does not require a registry rebuild.

To clear the theme registry, do one of the following things:

1. Clear button located at "Administer > Site configuration > Performance".
2. With devel block enabled (*comes with devel module*), click the "Empty cache" link.
3. The API function [drupal_rebuild_theme_registry](#).

The theme registry is cached data instructing Drupal on the available theming hooks and how to handle it by indicating its type. In previous versions all theming calls were handled on the fly. Since a lot more work is being done under the hood, the cached instructions speeds up the process especially for templates. The theme engine your theme is running under should automatically register all the theming hooks for you.

There are special cases where you may need to work with the registry directly. When your theme requires a new hook to be registered that was not already implemented on the layers below it (*core, modules, engine*). This includes some forms when they are not explicitly themed by core or modules but instead rely on the default form presentation.

- More details can be found in the sub-page, [The theme registry for special cases](#).
- Do not confuse the theme registry with the [theme's .info](#) file which is also cached. Points 1 and 2 on clearing the registry will clear both.

Preprocess functions

Preprocess functions only apply to theming hooks implemented as templates. The main role of the preprocessor is to setup variables to be placed within the template (`.tpl.php`) files. Plain theme functions do not interact with preprocessors.

Notes:

- Preprocessors are also used for providing [template suggestions](#).
- In versions 5 and below, the function `_phptemplate_variables` served the same purpose. It has been deprecated in 6.

There can be numerous preprocessors for each theming hook. Every layer from core, modules, engines and themes can have one, each progressively building upon the variables before being rendered through template files. This keeps the markup clean and easy to work with inside templates by placing most of the logic inside these preprocessors.

Here are the expected preprocessors. They are run in this order when they exist:

1. [template_preprocess](#)
 - This is supplied by core and always added. The [variables generated here](#) are used for every templated hook.
2. `template_preprocess_hook`
 - The module or core file that implements the theming hook supplies this. The initial generation of all the variables specific to the hook is usually done here.
3. `moduleName_preprocess`
 - Do not confuse this with the preprocessor before it. This allows modules that did not originally implement the hook to influence the variables. This would run for all hooks.
4. `moduleName_preprocess_hook`
 - Same idea as the previous preprocessor but for specific hooks.
5. `engineName_engine_preprocess`
 - The preprocessor for theming engines. Applies to all hooks.
6. `engineName_engine_preprocess_hook`
 - Another preprocessor for theming engines but specific to a single hook.
7. `engineName_preprocess`
 - This is the first preprocessor that can be used inside the theme. It can be named after the theme engine the theme is running under. Applies to all hooks.
8. `engineName_preprocess_hook`
 - Another preprocessor named after the engine but specific to a single hook.
9. `themeName_preprocess`
 - This one is named after the theme itself. Applies to all hooks.

10. `themeName_preprocess_hook`
- Same as the previous preprocessor but for a specific hook.

There are many possibilities here for modifying the variables. In most cases, it is only the first two preprocessors that exists. The first, adding the default baseline variables and the second, adding a set specific to the theming hook. Contributed modules taking advantage of the preprocessor slots (3 & 4) should document their behavior. It will not be covered extensively here.

While it is possible, the default PHPTemplate does not inject itself to this list. (5 & 6)

Themes can start adding their preprocessors seventh in the list. Although impractical, it is possible to grow this list beyond the ten shown above through sub-theming by adding strictly through the theme name as it is shown in the last two examples.

A few notes:

- It is recommended that base themes use the engine name (7 & 8) for its preprocessor. This makes it easier to move around code between themes and post snippets on [Drupal.org](#).
- The theme name (9 & 10) should **always be used for sub-themes**. This minimizes any chance of fatal errors due to duplicate functions which are illegal in PHP.
- In order for your theme to have its preprocessors recognized, the template associated with the hook must exist inside the theme. When a default template exists, copy it to your theme and clear the registry. If you are converting a theme function into a template, read the parent page on getting the [hook to register as a template](#).

Note that nothing should be [returned](#) from these functions and the variables have to be passed by [reference](#) indicated by the ampersand before variables, e.g., `&$variables`.

Since the variables set early on are run through all the latter preprocessors due to references, be careful that you do not inadvertently reset any added before your theme. It is fine to reset them but doing so accidentally can keep you guessing on what went wrong.

Example set from a module implementing the hook of "foo":

```
<?php
function template_preprocess_foo(&$variables) {
  $variables['foo_list'] = array(
    'list item 1',
    'list item 2',
    'list item 3',
  );
}
?>
```

And the preprocessor created from the theme to add to the variable set above:

```
<?php
function drop_preprocess_foo(&$variables) {
  // Do not do this unless you mean to:
  $variables['foo_list'] = array('list item 4');

  // Instead do this:
  $variables['foo_list'][] = 'list item 4';
}
?>
```

The variables that end up in the template file are the keys set within `$variables`. So, with the above example, the variable in the template would result in `$foo_list`.

Default baseline variables

The following are the baseline variables available to all [template files](#). They are generated through the [preprocessor function](#), [template_preprocess](#). Variables specific to the template are documented inside the file.

- `$id`
The placement of the template. Each time the template is used, it is incremented by one.
- `$zebra`
Either "odd" or "even". Alternate each time the template is used.
- `$directory`
The theme path relative to the base install. example: "sites/all/themes/myTheme"
- `$is_admin`
Boolean returns TRUE when the visitor is a site administrator.
- `$is_front`
Boolean returns TRUE when viewing the front page of the site.

\$logged_in

Boolean returns TRUE when the visitor is a member of the site, logged in and authenticated.

\$db_is_active

Boolean returns TRUE when the database is active and running. This is only useful for [theming in maintenance mode](#) where the site may run into database problems.

\$user

The user object containing data for the current visitor. Some of the data contained here may not be safe. Be sure to pass potentially dangerous strings through [check_plain](#).

The theme registry for special cases

You should be familiar with the [purpose](#) of the theme registry before continuing on this page. The instructions here will cover how to manually register a theming hook and explain how it can be [manipulated](#).

The most common case for manually registering hooks are in forms. Form elements are themable but there is another dimension to how they are processed. There are the generic elements such as checkboxes, radio selects, submit button, drop down menus, etc. Each element is themable on its own and overriding them does not require manually registering the hooks associated with them. The extra dimension comes in with custom forms where each element is arranged in very specific ways. In some forms, they are already arranged, themed and registered. For these forms, manually registering is not required. The forms that are not explicitly themed will default to how [form API renders](#) them.

Registered form example:

Here's an example for two of the search forms registered by search.module, the search box and search block. Every form has a unique id associated with them. Registering the id's also serve as the theming hook. In this case, it is "search_theme_form" and "search_block_form".

```
<?php
function search_theme() {
  return array(
    'search_theme_form' => array(
      'arguments' => array('form' => NULL),
      'template' => 'search-theme-form',
    ),
    'search_block_form' => array(
      'arguments' => array('form' => NULL),
      'template' => 'search-block-form',
    ),
    ...
  );
}
?>
```

Form API passes control of its presentation along to the handler of the registered hook. In this example, it is registered with the type of template with its default arguments. The output can be altered easily from the theme since it was already registered as a [template](#). Auto discovery of the hook only happens with existing theming hooks registered below the theming layer. ([see image](#))

Registering an unregistered form:

There is another search form that is not registered. It is the form with the id of "search_form" used on the main search page. The data for the form is [constructed in a function](#) just like any other form but it leaves it up to form API to deal with the presentation based on its data structure.

To extend this so it can be overridden, you have to register it from your theme with [hook_theme](#). Place the following inside your template.php file replacing the "drop" prefix with the name of your theme. The theming hook is the id of the form:

```
<?php
function drop_theme() {
  return array(
    'search_form' => array(
      'arguments' => array('form' => NULL),
    ),
  );
}
?>
```

Themed forms always have an argument of "form". Since the template was not specified, the hook will be seen as a theme function, not a template. *The theme function has to have a matching prefix of the theme that registered it. phptemplate_* will not work.* So, with the above entry, the theme function would look like this:

```

<?php
function drop_search_form($form) {
  $simple = '';
  foreach (element_children($form) as $element) {
    if ($element == 'advanced') {
      $advanced = drupal_render($form[$element]);
    }
    else {
      $simple .= drupal_render($form[$element]);
    }
  }
  return $advanced . $simple;
}
?>

```

The only thing changed here is the positioning of the advanced search form.

- *Sub-themes overriding a form that was registered in its base theme does not have to manually re-register the form. Remember, the registering allows auto discovery for the layers above it and sub-themes are just another layer on top of base themes.*
- *This may be more automated in future versions. The developers are aware of the burden it places on themers.*

Manual manipulating

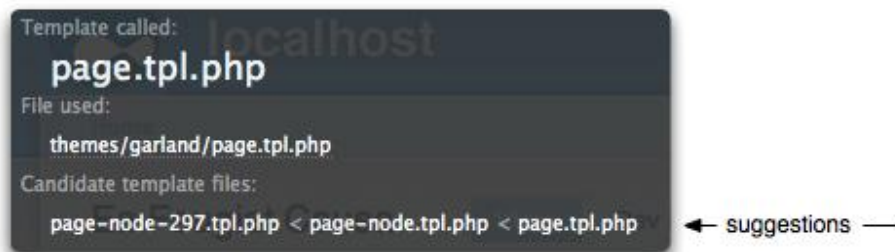
Manual manipulation of the theme registry is an advanced feature. You can read it by clicking the "Theme registry" in the block provided by devel module. It can also be returned with [theme_get_registry](#).

to be continued...

Working with template suggestions

Template suggestions are alternate templates based on existing .tpl.php files. These suggestions are used when a specific condition is met and a matching file exists. All layers from core, modules, theme engines and themes can provide the suggestions. You can think of them as *naming hints* telling the system to pick and choose based on the right circumstances. The idea is simple but it is a powerful feature providing another layer of customization.

Devel module showing template suggestions for the [possible "page" templates](#).



A listing of all the suggestions for core can be found in [Core templates and suggestions](#).

These naming suggestions are set from [preprocess functions](#). There are plenty already provided by core. If you need to extend it further, add a preprocessor for the theming hook into your template.php file. This example add suggestions on the "page" theming hook. It can be added to any hook implemented as a template.

The prefix of "drop" should be the name of your theme.

```

<?php
function drop_preprocess_page(&$variables) {
  global $user;

  // Add a single suggestion.
  if (module_invoke('throttle', 'status') && isset($user->roles[1])) {
    $variables['template_file'] = 'page-busy';
  }

  // Add multiple suggestions.
  if (!empty($user->roles)) {
    foreach ($user->roles as $role) {
      $filter = '!^[^abcdefghijklmnopqrstuvwxyz0-9-_-]+!s';
    }
  }
}

```

```

$string_clean = preg_replace($filter, '-', drupal_strtolower($role));
$variables['template_files'][] = 'page-'. $string_clean;
}
}
}
?>

```

There are two ways to add these suggestions.

1. The key of 'template_file' accepts a single suggestion and it takes precedence. If the condition is met and the file exists, it will be used ignoring all others.
2. The key of 'template_files' (*plural*) can accept an array of suggestions. They are processed in FILO order (*first in last out order*). Adding to the array should be done with a general condition first, progressively getting more specific so it cascades based on specificity.

With the above example, Drupal will attempt to use a file named "page-busy.tpl.php" when the throttling threshold is met for anonymous users (*anonymous role id typically set to 1*). The others inform Drupal to look for templates based on the roles assigned to the current user, e.g., "page-authenticated-user.tpl.php". If none apply, the base template of "page.tpl.php" is used.

These are simply examples. You can set any context based on any data available to you.

A few notes:

- When adding to 'template_files', add to the array. Do not reset it since the variables are passed by reference. All the suggestions set before it in core and modules would be lost.

```

<?php
// Do not do this:
$variables['template_files'] = array('hook-suggestion');

// Instead do this:
$variables['template_files'][] = 'hook-suggestion';
?>

```

- Prefix the suggestion with the name of the hook it is associated with. This keeps it clear and the files grouped together. It also minimizes any chance of Drupal registering the template with a different hook.
- Use hyphens instead of underscores for consistency. The main template will never use underscores.
- Suggestions work only when it is placed in the same directory as the base template. Templates can be placed in any sub-directory of the theme. They must be paired into the same location.
- The theme registry does not have to be cleared for suggestions. It's only the base template that needs to be registered. Suggestions are discovered on the fly.

Core templates and suggestions

Default templates:

These are the default template (*.tpl.php*) files provided by core. Documentation on the variables and purpose of these templates are located inside the templates. There is a [default set of variables](#) available to all templates.

The templates handled by PHPTemplate in 5.x were also moved. PHPTemplate no longer handles template files.

In order to override these templates, all you need to do is copy them into your theme folder and [clear the theme registry](#).

Aggregator

"modules/aggregator/..."

- [aggregator-feed-source.tpl.php](#)
- [aggregator-item.tpl.php](#)
- [aggregator-summary-item.tpl.php](#)
- [aggregator-summary-items.tpl.php](#)
- [aggregator-wrapper.tpl.php](#)

Block

"modules/system/..."

- [block.tpl.php](#)

"modules/block/..."

- [block-admin-display-form.tpl.php](#)

Book

"modules/book/..."

- [book-all-books-block.tpl.php](#)
- [book-export-html.tpl.php](#)
- [book-navigation.tpl.php](#)
- [book-node-export-html.tpl.php](#)

Comment

"modules/comment/..."

- [comment-folded.tpl.php](#)
- [comment-wrapper.tpl.php](#)
- [comment.tpl.php](#)

Forum

"modules/forum/..."

- [forum-icon.tpl.php](#)
- [forum-list.tpl.php](#)
- [forum-submitted.tpl.php](#)
- [forum-topic-list.tpl.php](#)
- [forum-topic-navigation.tpl.php](#)
- [forums.tpl.php](#)

Node

"modules/node/..."

- [modules/node/node.tpl.php](#)

Poll

"modules/poll/..."

- [poll-bar-block.tpl.php](#)
- [poll-bar.tpl.php](#)
- [poll-results-block.tpl.php](#)
- [poll-results.tpl.php](#)
- [poll-vote.tpl.php](#)

Profile

"modules/profile/..."

- [profile-block.tpl.php](#)
- [profile-listing.tpl.php](#)
- [profile-wrapper.tpl.php](#)

Search

"modules/search/..."

- [search-block-form.tpl.php](#)
- [search-result.tpl.php](#)
- [search-results.tpl.php](#)
- [search-theme-form.tpl.php](#)

User

"modules/user/..."

- [user-picture.tpl.php](#)
- [user-profile-category.tpl.php](#)
- [user-profile-item.tpl.php](#)
- [user-profile.tpl.php](#)

System

"modules/system/..."

- [page.tpl.php](#)
- [maintenance-page.tpl.php](#)
- [box.tpl.php](#)

Template suggestions:

Suggestions only work when it is placed in the same directory as the *base template*. In other words, trying to get **comment-blog.tpl.php** to work, *comment.tpl.php* also needs to exist inside your theme and in the same directory.

Custom suggestions beyond the ones listed below can be created. See the page [Working with template suggestions](#).

block-[region|[module]-delta].tpl.php

base template: *block.tpl.php*

Suggestions made based on these factors in this order:

1. *block-module-delta.tpl.php*
2. *block-module.tpl.php*
3. *block-region.tpl.php*

"module" being the name of the module and "delta", the internal id assigned to the block by the module. For example, "block-user-1.tpl.php" would be used for the default user navigation block since it was created by the user module with the id of 1. "region" will take effect for specific regions.

comment-[type].tpl.php

base template: *comment.tpl.php*

Support was added to create *comment-type.tpl.php* files, to format comments of a certain node *type* differently than other comments in the site. Similar to *node-[type].tpl.php* but for comments.

comment-wrapper-[type].tpl.php

base template: *comment-wrapper.tpl.php*

Similar to the above but for the wrapper template.

forums-[[container|topic]-forumID].tpl.php

base template: *forums.tpl.php*

Template suggestions added based in these factors, in this order.

For forum containers:

1. *forums-containers-forumID.tpl.php*
2. *forums-forumID.tpl.php*
3. *forums-containers.tpl.php*

For forum topics:

1. *forums-topics-forumID.tpl.php*
2. *forums-forumID.tpl.php*
3. *forums-topics.tpl.php*

maintenance-page-[offline].tpl.php

base template: *maintenance-page.tpl.php*

This applies when the database fails. Useful for presenting a friendlier page without error messages. [Theming the maintenance page](#) must properly setup first.

node-[type].tpl.php

base template: *node.tpl.php*

Node content type, e.g., "node-story.tpl.php", "node-blog.tpl.php", etc.

page-[front|internal/path].tpl.php

base template: *page.tpl.php*

The suggestions are numerous. The one that takes precedence is for the front page. The rest are based on the internal path of the current page. Do not confuse the internal path to path aliases which are not accounted for. *Keep in mind that the commonly used Path auto module works its magic through path aliases.*

The front page can be set through "Administrator > Site configuration > Site information". Anything set there will trigger the suggestion of "page-front.tpl.php" for it.

The list of suggested template files in order of specificity based on internal paths. One suggestion is made for every element of the current path, though numeric elements are not carried to subsequent suggestions. For example, "<http://www.example.com/node/1/edit>" would result in the following suggestions:

1. page-node-edit.tpl.php
2. page-node-1.tpl.php
3. page-node.tpl.php
4. page.tpl.php

poll-results-[block].tpl.php

base template: *poll-results.tpl.php*

The theme function that generates poll results are shared for nodes and blocks. The default is to use it for nodes but a suggestion is made for rendering them inside block regions. This suggestion is used by default and the template file is located inside "modules/poll/poll-results-block.tpl.php".

poll-vote-[block].tpl.php

base template: *poll-vote.tpl.php*

Similar to poll-results-[block].tpl.php but for generating the voting form. You must provide your own template to for it to take effect.

poll-bar-[block].tpl.php

base template: *poll-bar.tpl.php*

Same as poll-vote-[block].tpl.php but for generating individual bars.

profile-wrapper-[field].tpl.php

base template: *profile-wrapper.tpl.php*

The profile wrapper template is used when browsing the member listings page. When browsing specific fields, a suggestion is made with the field name. For example, <http://drupal.org/profile/country/Belgium> would "suggest profile-wrapper-country.tpl.php".

search-results-[searchType].tpl.php

base template: *search-results.tpl.php*

search-results.tpl.php is the default wrapper for search results. Depending on type of search different suggestions are made. For example, "example.com/search/node/Search+Term" would result in "search-results-node.tpl.php" being used. Compare that with "example.com/search/user/bob" resulting in "search-results-user.tpl.php". Modules can extend search types adding more suggestions of their type.

search-result-[searchType].tpl.php

base template: *search-result.tpl.php*

The same as above but for for individual search results.

Style sheets

The following will cover how Drupal handles style sheets through [.info files](#). See the sub-page on adding through the [API functions](#) for more advanced functionality. Styling themes purely through CSS is possible with the information provided here.

There are a few points to be aware of. Each core component or module will provide a reasonable default for its presentation. This includes the markup and an associated style sheet. (*see the explanation on the [overriding behavior for the markup](#).*) Due to the extensible nature of Drupal, it would be a great burden for themers to handle everything being pushed out to the browser. These defaults are there to be changed at the themers discretion. Just like the overriding behavior for themable functions and templates, the style sheets provided by core and modules can be overridden. Do not modify directly. All changes should be localized inside your theme.

Notes:

- *When working with style sheets, make sure CSS Optimization is disabled. It is located in "Administer > Site configuration > Performance". When it is enabled, any alterations will not be reflected on your site until the aggregated styles are cleared. You can enable it again when you're done.*
- *The .info file is cached. Adding or removing any styles will not be noticed until it is cleared. (Do not confuse this with the [theme registry](#).) To have it cleared, do one of the following:*
 1. *Clear button located at "Administer > Site configuration > Performance".*
 2. *With devel block enabled (comes with devel module), click the "Empty cache" link.*
 3. *Simply visit the theme select page at "Administer > Site building > Themes".*

Adding style sheets:

By default, a "style.css" file will be used from your theme when no other styles are defined inside the .info file. Adding

other styles is as simple as defining a new 'stylesheets' key with its [media property](#) and the name of the style sheet. Keep in mind that defining custom styles will prevent the default "style.css" from loading. Remember to explicitly define the default if your theme uses it.

```
; Add a stylesheet for all media
stylesheets[all][] = theStyle.css
```

```
; Add a stylesheets for screen and projector media
stylesheets[screen, projector][] = theScreenProjectorStyle.css
```

```
; Add a stylesheet for print media
stylesheets[print][] = thePrintStyle.css
```

A few notes:

- Note the empty square brackets between the [media] and = styleName.css.
- The order in which the styles are listed in the head of the page will reflect the order it is defined here.
- The style sheets can be placed in sub-directories, i.e., stylesheets[all][] = stylesheets/styleName.css. Useful for organizing style sheets.

Overriding core and module style sheets:

To override a core or module style sheet, it can be redefined from the themes .info file. Take "system-menus.css" as an example. It is located in "modules/system/system-menus.css". With this entry, the original CSS file will be ignored and your local copy will be loaded.

```
stylesheets[all][] = system-menus.css
```

When overriding a module style the file should be present inside your theme and pointed to the right location with a matching name. This ensures the original is replaced with the style sheet provided by the theme.

Adding the override for a style sheet that does not exist inside the theme will omit the core or module style sheet. This is by design and this behavior has been corrected since the release of 6.0 in 6.3.

A few notes:

- Overriding a core CSS files will prevent the default "style.css" file from loading. Remember to explicitly define any [defaults](#) when needed.
- The themes override must have a matching media type of the original style.
- Any linked elements within the style will have to be corrected. Double check the file for any 'url()' properties or '@import' rules to make sure they are pointing to the right resource.
- The order the style sheets listed in the head of the page will change. In effect, this may cause some cascading rules to change with it.
- Some core and module style sheets are loaded conditionally. Overriding through .info files will force the file to be always used.
- If the changes from the theme is not substantial, then consider using CSS selectors to override the styles instead of overriding the whole file.

Overriding the base theme's style sheets:

The following applies to [sub-themes](#). To prevent a style sheet from a base theme from being carried over to a sub-theme, you can redefine the style sheet inside the .info file. This works the same way as overriding module or core style sheets.

The base theme and the sub-theme must have the same entry:

```
stylesheets[all][] = masterStyle.css
```

If the file exists inside the sub-theme, then it will be used while omitting the file will prevent it from loading.

Adding styles through the API

Adding [styles through the .info file](#) should be sufficient for most themes. Since the .info file is static, style sheets cannot be added dynamically. Depending on how the theme handles style sheets, it may not matter altogether. When in doubt, use the .info file.

There are two API functions for working with style sheets, [drupal_add_css](#) and [drupal_get_css](#). Here is an example to dynamically add styles sheets.

Change the "drop" prefix to the name of your theme.

```

<?php
function drop_preprocess_page(&$variables) {
  $front_style = path_to_theme() .'/front-page.css';
  $path_style = path_to_theme() .'/path-' . arg(0) .'.css';

  if (file_exists($front_style) && $variables['is_front']) {
    $include_style = $front_style;
  }
  elseif (file_exists($path_style)) {
    $include_style = $path_style;
  }

  if (isset($include_style)) {
    drupal_add_css($include_style, 'theme', 'all', FALSE);
    $variables['styles'] = drupal_get_css();
  }
}
?>

```

The above example would include the style sheet "front-page.css" on the front page or many others based on the internal path. For example, <http://example.com/admin> would pickup on "path-admin.css".

A few notes:

- Depending on where and when the style is added, `drupal_get_css` may need to be called in order to include the added styles. They are initially retrieved in [template preprocess page](#). See [Preprocessors and variables](#) for details on the order of the preprocessors.
- There is a parameter in `drupal_add_css` to aggregate the added file. Consider disabling it like the above example when the inclusion of the style sheet is very dynamic since files added to the larger aggregate will force a new aggregated CSS file to be recreated. In effect, it can slow down the retrieval of the page and consume more bandwidth.

Overriding core stylesheets

By default Drupal comes installed with some things styled. While this is handy there are times when you are working on a theme that you want to change these settings for your design.

In this chapter there are some of the most common overrides people run into.

(Please fill this in as much as possible.)

Custom ul list-style

Difficulty level: intermediate to advanced.

Example 1: The basics

For this example we will create a unordered list(ul) that will show a disc next to a list item(li) only in a particular area, and will hide them in the rest of the website. You can then decide which lists you want to style in your website.

Note:

Make sure you do this in a development website, and in any case have a backup of the files you are going to work on.

We will take a div and name it "content". The name of your div may be different. In it we will place the ul list to style:

```

<div id="content">
<ul class="unordered_list_in_content">
<li>list item 1</li>
<li>list item 2</li>
<li>list item 3</li>
</ul>
</div>

```

In your stylesheet you would use:

```

ul { list-style-type:none } /* This will turn off all list-style-types in your theme */

#content ul.unordered_list_in_content { list-style-type: disc }

```

You should now see your list with discs in the area that you chose, and not in the rest of the website.

Example 2: Let's have some fun with lists styles

To turn off all list-style in a theme use:

```
ul {list-style:none}
```

instead of

```
ul {list-style-type:none}
```

You can do do something like this:

```
ul {list-style:disc inside}
```

instead of

```
ul {list-style-type:disc}
```

So your final stylesheet would look like:

```
ul {list-style:none}
```

```
#content ul.ordered_list_in_content { list-style: disc inside}
```

Note: If you have more than one stylesheet in your theme you will have to make sure there are no list-styles overriding your css, otherwise it still might not show.

Note: list-style-type has become deprecated, so you should read <http://www.w3.org/TR/html401/struct/lists.html> for details.

Tip: This works also for unordered lists in a node item such as a page or story.

Supporting RTL languages

Adding support for RTL (*Right to Left*) languages involves overriding the lateral styles through cascades and naming the file based on the style sheet it is paired to. The inclusion of the RTL style sheet is automated. The inclusion of the file depends on the language settings set for the site.

For example, in the core theme Garland, "style.css" is the main style sheet. It also includes "style-rtl.css" for right to left languages like Arabic or Hebrew. The inclusion of the two styles always loads with the main style first and the RTL style second. This allows cascading of all the rulesets within the two files without having to worry about specificity in the selectors used in the RTL style.

There is a coding standard to keep the rules organized. Rules that are dependent on the lateral positioning or dimensions should be commented with */* LTR */* indicating that the property is specific to a left to right layout. This includes floats, margins, padding, etc. Inline text should flow automatically as long as the theme sets the language direction of the document through the "page.tpl.php" template.

Example base style:

```
ul.primary-links {
  margin-top: 1em;
  padding: 0 1em 0 0; /* LTR */
  float: left; /* LTR */
  position: relative;
}
```

Corresponding RTL style:

```
ul.primary-links {
  padding: 0 0 0 1em;
  float: right;
}
```

While working with the main CSS file, this makes it easier to spot where changes may be needed in the RTL style.

Note that if your theme [overrides a module style](#), the associated RTL style will be omitted unless it is present in your theme.

JavaScript & jQuery

jQuery basics

As of version 6 of Drupal, jQuery 1.2.3 is included in Drupal core,

Default JavaScript file

Similarly to style.css, there is a new automatically included file named script.js for adding JavaScript code to a theme. The file should be placed in the theme's home directory.

JavaScript theming

There is now a theming mechanism for JavaScript code. Together with the automatically included script.js, this allows theme developers more freedom in the domain of scripted events on Drupal sites. Often, people's JavaScript code produces markup that is inserted into the page. However, this may contain HTML that has been hard-coded into the script, which did not allow alteration of the inserted code.

Modules provide default theme functions in the `Drupal.theme.prototype` namespace. For example, if I wanted to make my theme function `powered` (to display a "powered by Drupal" icon), it would look like this

```
Drupal.theme.prototype.powered = function(color, height, width) {  
  return '';  
}
```

This takes advantage of plain JavaScript. The above function would go in a theme's JavaScript file, avoiding the need for altering the related module's JavaScript files.

JavaScript theme functions are entirely free in their return value. They can vary from simple strings to complex data types like objects, arrays, and jQuery elements. Refer to the original (default) theme function to see what your custom theme function should return.

Still needed: jQuery intro, how to add aggregation (patch at <http://drupal.org/node/119441>) etc...

Blocks, content and their regions

The block regions available to the theme are defined within [.info files](#). It must be specified with the key of 'regions' followed by the internal "machine" readable name *in square brackets* and the human readable name as the value, e.g., `regions[theRegion] = The region label`. If none are defined, the following values are assumed.

```
regions[left] = Left sidebar  
regions[right] = Right sidebar  
regions[content] = Content  
regions[header] = Header  
regions[footer] = Footer
```

Keep in mind that the internal names are converted into region variables inside the "page.tpl.php" template automatically. In the above example, the `[left]` region will output all the blocks assigned to it through the `$left` variable. There are a few restrictions on naming [variables in PHP](#) so make sure the internal names conform to the same restrictions. It can only contain alphanumeric characters and underscores but it cannot start with a number.

The human readable name outside the square brackets are used for labeling the region in the block administration page located at "Administer > Site building > Blocks".

Here is the block administration table for Garland:

Block	Region	Operations
Left sidebar		
+ Navigation*	Left sidebar	configure
Right sidebar		
+ User login	Right sidebar	configure
Content		
+ Recent blog posts*	Content	configure
Header		
+ Syndicate*	Header	configure
Footer		
+ Powered by Drupal	Footer	configure

A few notes:

- There are [template \(.tpl.php\) files](#) available for rendering individual blocks.
- Adding a custom region prevents the defaults from being used. If you want to keep the defaults in addition to custom regions, manually add in the defaults.
- The order in which the regions are defined will be reflected in the block configuration table. Garland for example uses the default regions. Notice the order of the regions listed in the image.
- The contents of the .info file is cached in the database so altering it will not be noticed by Drupal. (Do not confuse this with the [theme registry](#).) To clear it, do one of the following:
 1. Clear button located at "Administer > Site configuration > Performance".
 2. With devel block enabled (comes with devel module), click the "Empty cache" link.
 3. Simply visit the theme select page at "Administer > Site building > Themes".

Upgrade notes:

- In Drupal 5 and below, regions were declared with `ThemeName_regions()` or `EngineName_regions()`. It has been deprecated in Drupal 6.
- If you are upgrading your theme from versions before Drupal 6 and the region variables of `$sidebar_left` and `$sidebar_right` were used, rename them to `$left` and `$right`.
- The `$footer_message` region variable in versions before 6 mixed the footer region with the footer message (set from "Administer > Site configuration > Site information"). Make sure a separate `$footer` variable is created if your theme uses it since Drupal 6 and above no longer combines the two elements.

Manually assigning content to regions:

Content can be manually set inside regions with [drupal_set_content](#). For example, `drupal_set_content('header', 'Welcome!')` would assign the text 'Welcome!' to the header region.

Here is a more useful example for building a summary of all the comments into the "right" region. Rename the "drop" prefix with the name of your theme. More information on [preprocessors is available](#).

```
<?php
function drop_preprocess_comment(&$variables) {

  // Setup a few variables.
  $comment = $variables['comment'];
  $title = l(
    $comment->subject,
    comment_node_url(),
    array('fragment' => "comment-{$comment->cid}")
  );
}
```

```

$new_marker = $comment->new ? t('new') : "";
$by_line = t('by') . ' ' . theme('username', $comment);

// Form the markup.
$summary = '<div class="comment-sidebar">';
$summary .= "<span class='title'>$title $new_marker</span>";
$summary .= "<span class='credit'>$by_line</span>";
$summary .= '</div>';

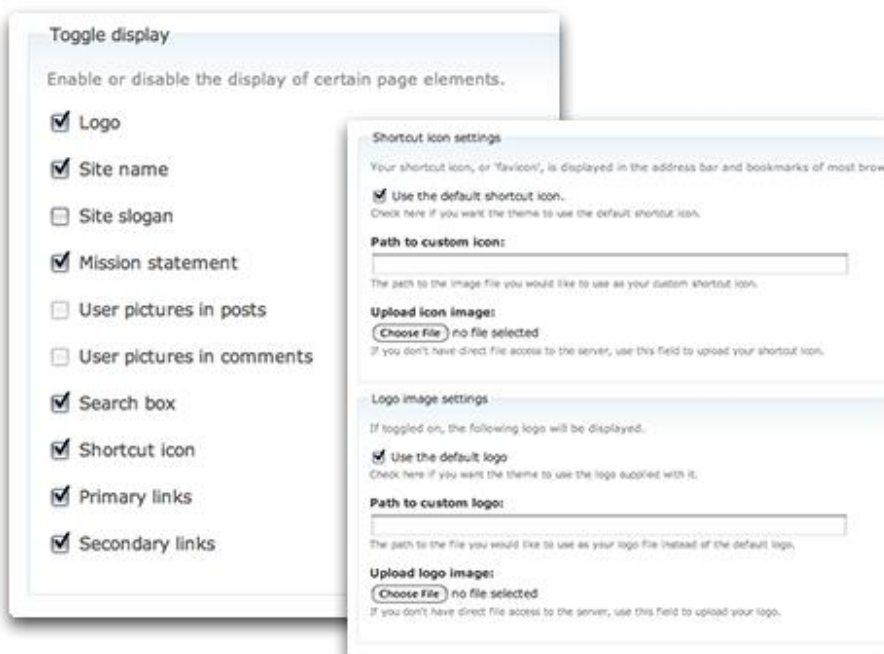
// Set the comment into the right region.
drupal_set_content('right', $summary);
}
?>

```

Note that setting content through this function should happen before the block regions are retrieved and that is done with a call from [template preprocess page](#) > [theme blocks](#) > [drupal_get_content](#).

Custom theme settings

Various page elements output by the theme can be toggled on and off on the theme's configuration page. Located at "Administer > Site building > Themes > themeName". For example, the site's slogan can be suppressed by unchecking the "Site slogan" check box on that page.



These checkboxes show themselves depending on the the features enabled inside the [.info file](#). It must be specified with the key of 'features' followed by empty brackets then the feature itself, e.g., features[] = the_feature. If none are defined, the following values are assumed.

```

features[] = logo
features[] = name
features[] = slogan
features[] = mission
features[] = node_user_picture
features[] = comment_user_picture
features[] = search
features[] = favicon
features[] = primary_links
features[] = secondary_links

```

To disable any features, only add the ones you want into the .info file. Defining only the features needed for the theme will omit the rest. Some of the features will also enable related form fields. For example, 'logo' will enable an upload field for the image along with the checkbox.

A few notes:

- The contents of the `.info` file is cached in the database so altering it will not be noticed by Drupal. (Do not confuse this with the [theme registry](#).) To clear it, do one of the following:
 - Clear button located at "Administer > Site configuration > Performance".
 - With devel block enabled (comes with devel module), click the "Empty cache" link.
 - Simply visit the theme select page at "Administer > Site building > Themes".
- `hook_features()` is no longer supported.

Advanced theme settings

In the Drupal administration section, each theme has its own settings page at `admin/build/themes/settings/themeName`. And this page has a form with standard settings like "Logo image settings" and "Shortcut icon settings."

In Drupal 6, theme authors can now customize this page by adding additional settings to the form. In Drupal 5, theme authors and theme users will have to install the [Theme Settings API](#) module (5.x-2.1 or later) before being able to use the method described below.

Adding form widgets for your custom theme settings

First, create a `theme-settings.php` file in your theme directory and add a `themeName_settings()` or `themeEngineName_settings()` function. The `themeEngineName_settings()` form is preferred since it allows others to more easily create derivative themes based on your theme. The function should use the Forms API to create the additional form widgets.

For example: to add settings to the Garland theme, a `garland_settings()` or `phptemplate_settings()` function would be placed in the theme's `theme-settings.php` file.

If a user has previously saved the theme settings form, the saved values will be passed to this function in the `$saved_settings` parameter. The widgets to add to the form should be returned as a Forms API array.

The comments in the following example explain the details:

```
<?php
// An example themes/garland/theme-settings.php file.

/**
 * Implementation of THEMEHOOK_settings() function.
 *
 * @param $saved_settings
 *   array An array of saved settings for this theme.
 * @return
 *   array A form array.
 */
function phptemplate_settings($saved_settings) {
  /*
   * The default values for the theme variables. Make sure $defaults exactly
   * matches the $defaults in the template.php file.
   */
  $defaults = array(
    'garland_happy' => 1,
    'garland_shoes' => 0,
  );

  // Merge the saved variables and their default values
  $settings = array_merge($defaults, $saved_settings);

  // Create the form widgets using Forms API
  $form['garland_happy'] = array(
    '#type' => 'checkbox',
    '#title' => t('Get happy'),
    '#default_value' => $settings['garland_happy'],
  );
  $form['garland_shoes'] = array(
    '#type' => 'checkbox',
    '#title' => t('Use ruby red slippers'),
    '#default_value' => $settings['garland_shoes'],
  );

  // Return the additional form widgets
  return $form;
}
?>
```

Note that theme authors can create complex, dynamic forms using advanced Forms API (auto-completion, collapsible fieldsets) and JQuery javascript.

Getting the settings' values in your theme files

In order to retrieve the settings in the theme's template.php or .tpl.php files, simply use `theme_get_setting('varname')`. See the Drupal API for details: http://api.drupal.org/api/6/function/theme_get_setting

For example:

```
<?php
$happy = theme_get_setting('garland_happy');
?>
```

Initializing the default values

Since we can't guarantee that a user will ever go to the **admin/build/themes/settings/themeName** page, we have to ensure that the default values for our custom settings get initialized.

The theme settings variables aren't set until we submit the **admin/build/themes/settings/themeName** form for the first time, so in our template.php file we need to check whether the variables are set or not. If they aren't set, we need to set them to the default values. We accomplish that by retrieving one of the variables and seeing if it is null. If it is null, we save the defaults using `variable_set()` and then force the refresh of the settings in Drupal's internals using `theme_get_setting("", TRUE)`.

Add the following code near the top of your template.php file:

```
<?php
/*
 * Initialize theme settings
 */
if (is_null(theme_get_setting('garland_happy'))) { // <-- change this line
  global $theme_key;

  /*
   * The default values for the theme variables. Make sure $defaults exactly
   * matches the $defaults in the theme-settings.php file.
   */
  $defaults = array( // <-- change this array
    'garland_happy' => 1,
    'garland_shoes' => 0,
  );

  // Get default theme settings.
  $settings = theme_get_settings($theme_key);
  // Don't save the toggle_node_info_ variables.
  if (module_exists('node')) {
    foreach (node_get_types() as $type => $name) {
      unset($settings['toggle_node_info_' . $type]);
    }
  }
  // Save default theme settings.
  variable_set(
    str_replace('/', '_', 'theme_' . $theme_key . '_settings'),
    array_merge($defaults, $settings)
  );
  // Force refresh of Drupal internals.
  theme_get_setting("", TRUE);
}
?>
```

Note that the variable name "garland_happy" in the first line of the above code would be replaced with a variable name from your custom theme settings and the \$defaults array would need to be copied from your theme-settings.php file.

Adding additional settings to a new version of your theme

After you have released a 1.0 version of your theme, you will eventually want to add some additional custom settings to the 2.0 version. The process is mostly straight-forward. But pay close attention to the initialization code in the third step:

1. In your theme-settings.php file, add the new settings to the \$defaults and \$form variables.
2. In your template.php file, add the settings to the \$defaults variable in the *Initialize theme settings* code.

3. In your `template.php` file, update the initialization code to check for the existence of one of your *new* settings. For example, if you added several settings, including a `garland_slippers` setting, you would change the first line of the *Initialize theme settings* code to read:
`if (is_null(theme_get_setting('garland_slippers'))) {`

This will ensure that the defaults for your newly-added custom settings get added to the saved values of the old custom settings.

Integrating color module

Color.module allows the admin to change the color scheme of a theme completely. By selecting a palette of 5 colors (either from a set or by hand), you can change the colors of an entire theme.

The module can alter the stylesheet and re-render images. However, the theme must provide specific hooks to allow this, and the design must be created specifically to accommodate this.

This document explains the basics of making a colorizable theme.

Design



It is important to realize that due to the way color.module works, not every design can be colorized.

We take a transparent image of the design (the base), which includes everything except the background. We then compose this image on top of a colored background, to get the colored versions. Finally we slice up this composite image into smaller images and save them to separate image files.

We also process the stylesheet and change all the colors based on the ones you defined. The module smoothly changes the colors using the palette as a reference. Colors that don't appear literally in the palette are adjusted relative to the closest matching palette color (whether it is a link, text or background color).

So, the photoshop mockup of the design should consist of a layered file that has one or more colored layers at the bottom of the layer stack, with everything else blended on top. When you save the base image, you have to merge all layers together, while keeping the colored layers invisible. Take a look at Garland's [base.png](#) file to see an example (open it in an image

editor to see the transparencies).

All the files generated in this process are written to /files/css and used instead of the default images. This means that a colorizable theme should still work out of the box without color.module, in the default color scheme.

In Practice

Let's use Garland as an example. The most important files are in the themes/garland/color subdirectory:

base.png

This contains the base design of the theme, which is composed and sliced into the images.

color.inc

This file contains all the necessary parameters to color the theme. See below.

preview.css

This stylesheet is used to generate the preview on the color changer.

preview.png

This image is used to generate the preview on the color changer.

The presence of color/color.inc makes the color picker appear on the theme's settings. It is a regular PHP file which contains an \$info array with the following values:

Schemes

```
<?php
array('schemes' => array(
  '#0072b9,#027ac6,#2385c2,#5ab5ee,#494949' => t('Blue Lagoon (Default)'),
  '#464849,#2f416f,#2a2b2d,#5d6779,#494949' => t('Ash'),
  '#55c0e2,#000000,#085360,#007e94,#696969' => t('Aquamarine'),
  '#d5b048,#6c420e,#331900,#971702,#494949' => t('Belgian Chocolate'),
  '#3f3f3f,#336699,#6598cb,#6598cb,#000000' => t('Bluemarine'),
  '#d0cb9a,#917803,#efde01,#e6fb2d,#494949' => t('Citrus Blast'),
  '#0f005c,#434f8c,#4d91ff,#1a1575,#000000' => t('Cold Day'),
  '#c9c497,#0c7a00,#03961e,#7be000,#494949' => t('Greenbeam'),
  '#ffe23d,#a9290a,#fc6d1d,#a30f42,#494949' => t('Mediterrano'),
  '#788597,#3f728d,#a9adbc,#d4d4d4,#707070' => t('Mercury'),
  '#5b5fa9,#5b5faa,#0a2352,#9fa8d5,#494949' => t('Nocturnal'),
  '#7db323,#6a9915,#b5d52a,#7db323,#191a19' => t('Olivia'),
  '#12020b,#1b1a13,#f391c6,#f41063,#898080' => t('Pink Plastic'),
  '#b7a0ba,#c70000,#a1443a,#f21107,#515d52' => t('Shiny Tomato'),
  '#18583d,#1b5f42,#34775a,#52bf90,#2d2d2d' => t('Teal Top'),
));
?>
```

This entry contains a straightforward array of pre-defined color schemes. Each entry must have 5 colors, formatted as above, and a title.

The first scheme is used as a reference and must match the colors used in the theme's default images and stylesheet closely. Otherwise, the final colors might not be what the user intended. See the 'stylesheets' section for more information about how the colors are calculated.

Images to copy

```
<?php
array('copy' => array(
  'images/menu-collapsed.gif',
  'images/menu-expanded.gif',
  'images/menu-leaf.gif',
));
?>
```

This array contains a list of images which should not be altered. They are copied to the location of the generated images and stylesheet.

Fill areas and Gradients

To color the image, we create a target image that is the same size as the base image, and draw colored areas and a gradient. For full flexibility, the location of these areas is defined by specifying their coordinates using (x, y, width, height):

```
<?php
array('gradient' => array(0, 37, 760, 121));
?>
```

You can specify one vertical two-color gradient.

```
<?php
array('fill' => array(
  'base' => array(0, 0, 760, 568),
  'link' => array(107, 533, 41, 23),
));
?>
```

You can specify regions for each of the palette colors. The region will be filled in with the selected color. Available colors are 'base', 'link', 'top', 'bottom' and 'text'.

Image slices

Next, you need to define the areas of the base image to slice out for each of the images. Again, you specify coordinates as (x, y, width, height) along with the filename of the image, as used in the stylesheet. The logo and screenshot slices are special and always take the same filename. The screenshot will be resized to 150x90 pixels.

```
<?php
array('slices' => array(
  'images/body.png' => array(0, 37, 1, 280),
  'images/bg-bar.png' => array(202, 530, 76, 14),
  'images/bg-bar-white.png' => array(202, 506, 76, 14),
  'images/bg-tab.png' => array(107, 533, 41, 23),
  'images/bg-navigation.png' => array(0, 0, 7, 37),
  'images/bg-content-left.png' => array(40, 117, 50, 352),
  'images/bg-content-right.png' => array(510, 117, 50, 352),
  'images/bg-content.png' => array(299, 117, 7, 200),
  'images/bg-navigation-item.png' => array(32, 37, 17, 12),
  'images/bg-navigation-item-hover.png' => array(54, 37, 17, 12),
  'images/gradient-inner.png' => array(646, 307, 112, 42),

  'logo.png' => array(622, 51, 64, 73),
  'screenshot.png' => array(0, 37, 400, 240),
));
?>
```

Files

Finally you need to specify the location of the files for your theme. You need an image and a stylesheet for the preview, as well as the base image*:

```
<?php
array(
  'preview_image' => 'color/preview.png',
  'preview_css' => 'color/preview.css',
  'base_image' => 'color/base.png',
);
?>
```

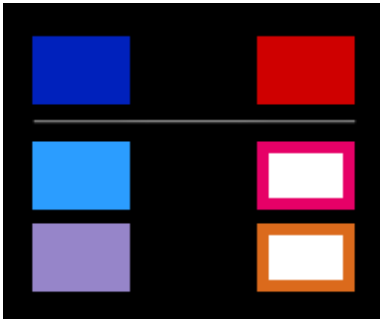
* As of drupal 6, Color.module will [no longer require base_image](#), meaning it is possible to utilize the module without images.

Stylesheets

The color.module will read in a theme's style.css file as well as any other styles that are imported with @import statements and create a new style.css file. It will change the colors in the CSS using one of the chosen palette colors as a reference, depending on the context:

- **Links:** the 'link' color is used, for rules that apply to a elements.
- **Text:** the 'text' color is used, for rules that appear in color: styles.
- **Base:** the 'base' color is used for everything else.

However, if a color in the stylesheet matches one of the reference colors exactly, the context will be ignored, and the matching replacement color will be used instead.



For example, suppose your reference color is *dark blue* by default, but you change it to *red*. Your default stylesheet contains both *light blue* and *gray purple*, both relative to this reference color.

The resulting colors (mauve and brown) are similarly different from red as the original colors were from blue. In technical terms: the relative difference in hue, saturation and luminance is preserved.

If you find `color.module` is using the wrong reference color, try separating the different pieces into separate CSS rules, each in their own selector `{ ... }` section, so there is no confusion about the context.

Note that if you edit your stylesheet after changing the color scheme, you must resubmit the color scheme to regenerate the color-shifted version.

If wish for certain colors in the stylesheet not to be altered, you should place their CSS below the following marker:

```

/*****
* Color Module: Don't touch
*****/

```

You can only use this marker once in your `style.css` file. It applies globally, so if you use it inside an imported stylesheet, all colors below the `@import` statement will be left alone too.

Making colors match

It is important that the generated images match with the shifted colors in the generated stylesheet. Otherwise, ugly edges might appear.

To make this work, pixels in the base image must all be a simple color in areas where they have to match with CSS-defined colors. Because we don't know where CSS-defined colors appear in the base image, we use a global blending color which must be the same in the whole design. Garland uses white. Note that the Garland base does include e.g. gray and black pixels, but only in areas where only images are used as backgrounds (e.g. the header). Other than white, black or grey are good candidates too.

```

<?php
array('blend_target' => '#ffffff');
?>

```

Masochists can take a peek at `color.module`'s innards, particularly the `_color_shift()` function if you're interested in the how and why of this.

PHPTemplate changes

Finally, you need to hook `color.module` into your theme. We'll use a PHPTemplate theme as an example, but this applies to other engines as well.

In your theme's `template.php` file, add the following snippet (for Drupal 6.x):

```

<?php
/**
 * Override or insert PHPTemplate variables into the templates.
 */
function phptemplate_preprocess_page(&$vars) {
  // Hook into color.module
  if (module_exists('color')) {
    _color_page_alter($vars);
  }
}
?>

```

In Drupal 5.x, you'll need to the following snippet:

```

<?php
/**

```

```

* Override or insert PHPTemplate variables into the templates.
*/
function _phptemplate_variables($hook, $vars) {
    if ($hook == 'page') {

        // Hook into color.module
        if (module_exists('color')) {
            _color_page_alter($vars);
        }
        return $vars;
    }
    return array();
}
?>

```

This will allow the module to override your theme's logo, stylesheet and screenshot. If you perform other changes in `_phptemplate_variables`, you need to merge in this snippet.

Theming the maintenance page

The maintenance page is used when the site is set into off-line mode. *You can enable this mode from "Administer > Site configuration > Site maintenance"*. This mode will also trigger with a database failure. By default, the core theme Minnelli is used in this mode even if another theme is selected. To use your theme for the maintenance page, it has to be set from your "settings.php" file located in either "sites/default" or "sites/your.domain.com".

From that file, enable it through the `$conf` variable with the internal name of your theme:

```

<?php
$conf['maintenance_theme'] = 'themeName';
?>

```

Next, copy your main "page.tpl.php" and rename it to "maintenance-page.tpl.php" or copy the template located in "modules/system/maintenance-page.tpl.php" to your theme and edit so it matches the rest of your site. Verify the changes by setting the site into off-line mode then log out.

To account for database failures, try turning off your database. Any function call that depends on the database should be checked first with [db_is_active](#). The variable `$db_is_active` can also be used from the template.

To prevent warnings about the database connection from appearing, you can also place a template file named "maintenance-page-offline.tpl.php" and change the `$content` variable with your custom message. This template file is a [template suggestion](#) based on maintenance-page.tpl.php so they both need to exist.

A "maintenance.css" file is included in this mode. It is located in "modules/system/maintenance.css". You can override this file with the instructions provided in the [style sheets section](#).

Notes:

- *The initial install and updating of your site depends on the core themes, Minnelli and Garland. It cannot be changed for these modes.*
- *The theme registry is not cached when set to off-line mode.*

Troubleshoot your theme

For all the effort you put into building your site the most important thing to your users is how your site looks. Managing the inconsistencies in your theme for each browser, for each module, and for user selectable themes can be very challenging.

First you should familiarize yourself with basic Cascading Style Sheets (CSS) concepts. Review [CSS Discuss](#) or [HTML dog](#) for CSS resources. A good overview of the power of CSS can be seen at the [CSS Zen Garden](#).

Validate! Always. Validate XHTML 1.0 Strict.

There is no way you can know what certain browsers do in certain quirky situations. Which causes the inconsistencies. But adding your load to that uncertainty, makes things harder and [impossible for others to help you](#). Only if you have validated XHTML and CSS and if it looks good in standard compliant browsers, you can start debugging. Too often one hears designers "butbutbut, if i make it validate, then it looks awfull again, I don't want to start all over again" . In that case, the fact that designer had a good looking layout, was due to a bug! debugging on top of bugs is just not an option!

And really: A validating, clean layout will work 90% of the time. will have easy fixes 9% of the time and will only require hard measures 1% of the time. Really!

If you would like to validate your whole site you can [use these tools](#).

Your page should not look the same everywhere

Another very important thing to notice is the nature of HTML and CSS. They are meant to look different in different places. My mobile phone is not able to show your fancy 6 column fixed with javascript based layout. And it should not be able to. Safari and Konqueror decided to not allow certain styles in forms (security and desktop consistency). Large screens will resize your font and that might break your fixed with layout. People using older screens will have larger font sizes set. People with bad internet connections often disable images. Or even CSS.

So keep in mind that your style is nothing more than an **advice** for browsers to display it in certain ways. It is by no means a **law**.

Tools for managing inconsistencies in your theme

1. We recommend you start with a standards compliant browser for your themes such as [Firefox](#). Firefox allows you to highlight portions of your web page and right click *view selected source* to understand what your themes CSS classes are. **Understanding how CSS classes for you theme are applied to the underlying XHTML is the key to understanding your theme.**
2. Use standard CSS naming conventions. We recommend adopting these [naming conventions](#) for you CSS classes.
3. [Select a valid DOCTYPE type](#) for your theme and [include a DocType Declaration\(DTD\)](#).
4. To help in analyzing your HTML and CSS we recommend you install the [Firebug plugin](#) for Firefox. It is an invaluable tool that allows you to **look at your css and html and change it in real time** to evaluate the effects of your changes. Another very useful Firefox plugin is the [Web Developer toolbar](#). This has tons of handy utilities.
5. The [view formatted source extension](#) for FireFox displays formatted and color-coded source and optional CSS information for each element.
6. Before you start modifying your CSS to fix your bugs it is important to ensure you are styling valid HTML or XHTML. There is a web validator built into the Firefox Web Developer toolbar. Opera has built in validation, just press Ctrl+Alt+V.
7. A more advanced tool for analyzing HTML pages for looking over code, spotting errors is the [Watchfire WebXACT tool](#).
8. If you find a problem with invalid XHTML in a module file an issue and include screenshots showing how this causes problems in the theme.
9. To see how your site will look to search engines you can use [Lynx viewer](#).
10. Positioning problems in your site with Internet Explorer can be resolved using these references: [Position Everything](#) [Internet Explorer Primer](#)
11. A library of examples of problems can be found at [Quirks mode](#).

Cross browser compatibility (FireFox, Internet Explorer, Opera, Safari)

It's hard to check your theme in all browsers. There are a number of tools which are available to help look at your theme in multiple browsers.

1. [Browser Shots](#) is freely available but can take a while to get your screenshots.
2. [BrowserCam](#) is paid service with a 24 hour trial.

On Windows you can use Internet Explorer and download Firefox or Opera. On Linux you can use Konqueror, a KHTML based browser, which Safari uses on MacOS, Opera, Firefox for Linux, and Internet Explorer can be run under WINE. On Mac OSX you can use Safari and download Firefox or Opera.

Color and graphic issues

1. If you are trying to select colors you can use [Color schemes](#).
2. If your theme is causing errors in your logs with favico not found you can create a favicon with these two tools [favicon from pics](#) and [favicon generator](#).
3. To check for how color blindness effects users see [Vischeck](#)

Selecting a base theme

If you are looking for a theme to start with [Zen](#) or [Foundation](#) are good base themes for CSS-based theming. There is also [BlueMarine](#), which uses tables for layout.

Module specific CSS

Some Drupal modules come with a default CSS file. You should use a tool like the developer's toolbar to see if a module's CSS is styling your elements and causing problems. When you install a new module you can also look in the module's folder to see if it is including a CSS file.

Actually debugging problems in your theme

There is no easy solution to debug your theme. If you are having trouble you should try to use the simplest base theme possible or select a theme that is known to work that is as close to your end goal as possible. Learning the CSS classes in

your theme is critical to understanding where your CSS changes will be applied. Try to find other themers by talking in IRC, providing documentation, and tutorials of what you have done. Make your existing theme available so that others can review and provide feedback on what you have done. Making friends with PHP programmers who can help you understand what the underlying PHP theme template is doing is also very important. Consider an exchange of skills to get support.

This page was written by Kieran Lal, and Trae McCombs from CivicSpace Labs with Theodore Serbinski. If you would like to contribute or help make Drupal theming easier join the [themes mailing list](#), or contact Kieran.

Basic theme help

Theme Goals

Improving your sites theme can help you to accomplish many goals. In particular, improving your site's theme can help with the following business goals:

- help drive traffic to your site
- highlight products for sale on your site
- provide information to your site visitors
- help your sites visitors to collaborate with each other.

This purpose of this themes help documentation is to help you accomplish the following theme development goals:

- make small design changes without having cross browser compatibility issues
- implement designers wireframes in your theme
- pick a base theme to customize
- build a new theme based on an existing theme

Improving theme help in Drupal

In a recent survey of Drupal administration user experience respondents indicated that theming was the most difficult Drupal administration task. We conducted a series of interviews to learn more about themers goals and the tasks that need to complete when developing a theme.

Basic Drupal theme tasks

Which ever theme you choose to work with you will need to know how to accomplish some basic theme tasks.

- Find and open a Cascading Style Sheet(CSS) file for your theme
- Copy and paste CSS code
- Learn the CSS attributes in your theme

It is recommended you use the Firefox browser with developer toolbar and the view formatted source extensions. We also recommend you use standard CSS ID names, as recommended by [Andy Clarke](#) and [Eric Meyer](#).

- Change colors
- Use part of another theme. For example, copy the CivicSpace Admin theme into your sites theme.
- How to build a new layout on top of an existing theme. E.g. change number of columns, header and footer positions, fluid versus static body.

Difficult theme tasks

We have identified a number of difficult theme tasks that require explanation. Some tasks are both basic tasks and difficult tasks and must be further documented to explain the difficult portions of these tasks.

- CSS Layout development
- Identifying a good base theme to start with

We have added categorization capabilities to the project module which will allow themes to be categorized.

- Alter an existing theme
- Learn CSS classes and IDs

It is recommended you use the Firefox browser with developer toolbar and the view formatted source extensions. We also recommend you use Eric Meyer's standard CSS class names.

- How to add padding or margins to a class
- Remove text and images from the theme UI. E.g. Remove submitted by, which may or may not be a theme issue.
- How to fix forms that are too difficult by default.
- Fix XHTML from modules that cause the theme to display incorrectly

An issue should be filed against the module. Include a picture of the incorrect display, how it looks in a basic theme like box grey. Provide a snippet of the offending XHTML from the module and indicate what the desired XHTML should look like.

- Working with PHP variables.
- Abandoning fixed width and working with containers
- Working with complicated fullheight designs or something flexible that needs lots of max-width and min-width to be done right.
- Inserting cool graphical effects designers want that will also work in Internet Explorer.
- Internet Explorer and Firefox compatibilities
- Write a theme from scratch

Theme coding conventions

Theme authors should take care to write clean, well structured code just like a coder for any other project. Doing so makes the code easier to read, understand and maintain. While different organizations have different conventions, it's usually best to follow the Drupal standards as this helps when collaborating or asking for help.

- Add 2 spaces for indents; rather than a tabbed indent
- Match the indentation of *long* opening and closing block HTML tags
- Distinguish between PHP and HTML indentation.

Not this:

```
...
<?php if ($header): ?>
<div id="header">
  <?php print $header; ?>
</div>
<?php endif; ?>
...
```

but this:

```
...
<?php if ($header): ?>
  <div id="header">
    <?php print $header; ?>
  </div>
<?php endif; ?>
...
```

This makes it much easier to find matching opening and closing tags defined with different indentation.

- Prefer PHP in HTML to HTML in PHP in templates. For example,

not this:

```
<?php
if (!$page) {
  print "<h2><a href=\"$node_url\" title=\"$title\">$title</a></h2>";
}

if ($submitted) {
  print "<span class=\"submitted\">$submitted</span>";
}
?>
```

but this:

```
<?php if (!$page): ?>
  <h2><a href="<?php print $node_url ?>" title="<?php print $title ?>"><?php print $title ?></a></h2>
<?php endif; ?>

<?php if ($submitted): ?>
  <span class="submitted"><?php print $submitted ?></span>
<?php endif; ?>
```

After all, PHP is a HTML embedded scripting language - and not the other way around.

Theme screenshot guidelines

Screenshots within a Drupal install

Every theme for 4.5+ needs a screenshot in the form of a screenshot.png placed in the theme directory. These are displayed in the theme listing within a Drupal installation (i.e. at Administer > Site building > Themes in Drupal 5.x or greater.) It is best that screenshots are consistent. The guidelines for core theme screenshots are (starting from a blank Drupal site):

1. Log in as the first user.
2. Enable the following modules, for some extra menu items: *aggregator*, *blog*, *node*, *page*, *search*, *story* and *tracker*.
3. Turn on the features that the theme supports (logo, site name, slogan, search box). Add some primary and secondary links if needed. We suggest "Link 1" "Link 2" "Link 3", you can link them to e.g. "user/1".
4. Set the site name to *Drupal* and slogan to *Community Plumbing*.
5. Create the following story node:

Donec felis eros, blandit non

Morbi id lacus. Etiam malesuada diam ut libero. Sed blandit, justo nec euismod laoreet, nunc nulla iaculis elit, vitae. Donec dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus vestibulum felis nec libero. Duis lobortis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nunc venenatis pretium magna. Donec dictum ultrices massa. Donec vestibulum porttitor purus. Mauris nibh ligula, porta non, porttitor sed, fermentum id, dolor. Donec eu lectus et elit porttitor rutrum. Aenean justo. Phasellus augue tortor, mattis nonummy, aliquam euismod, cursus eget, ipsum. Sed ultricies bibendum ante. Maecenas rhoncus tincidunt eros.

6. Look at the node, and make sure the tabs are visible. Take a screenshot.
7. Cut out a piece about ~420x254 resized to exactly 150x90 (~35% of the original size). Try to show only useful page elements (menu, tabs, title, links). Don't include browser chrome (toolbar, status bar, scrollbar, etc).
8. Apply a standard 'sharpen' filter to the thumbnail for clarity.
9. Save as a PNG, in paletted colorspace to cut down on size.

Example:



Screenshots for Drupal.org display

For Drupal.org project thumbnails (displayed at <http://drupal.org/project/Themes> and on the individual theme project pages), use the guidelines above except that:

- You should fill up the site more. For example, add a comment to the story node or add some blocks.
- The screenshot should show the entire page, though still without browser chrome (toolbar, status bar, scrollbar, etc).
- Try to make your original screenshot have a width of about 1000 pixels, so that the thumbnail is about 30% of the original size.
- Try to keep the image small: save as paletted PNG or as a JPEG with 10-20% compression. This reduces the load time of the theme list on Drupal.org.
- To add the image to your project, click the Edit tab on the project node and expand the "Attached Images" section. Browse to find your screenshot locally and add it. A thumbnail will be created and added to the upper right corner of the project page once you submit your changes.

If you need to change the image you have uploaded, click the thumbnail on your project page (this takes you to the image node) and then click the Edit tab. You may then upload a new picture which will replace the existing one.

Note that this method of thumbnail placement is quite different from the method listed here previously. That old method (linking to the CVS version) will no longer work. This new method does not require site admin intervention to add and change pictures.

Adding your theme to Drupal.org

To add your theme to Drupal.org, it must be [GPL](#)'d. Do not include images or other copyrighted works that you do not want to see re-used or otherwise altered.

Themes are tracked the same way that code is, in the CVS repository. You will need to [apply for a CVS account](#). Once you are approved, you will be able to check your theme into the Drupal CVS repository. Create a project and the download will be created for it automatically.

If you do add your theme, users will likely post suggestions, file bugs, and generally desire that you keep the theme up to date with current versions of Drupal.

Also read the [screenshot](#) guidelines.

You can find out more about the process of contributing code and themes and maintaining a project on Drupal.org [here in](#) the Developers handbook.