Converting 6.x themes to 7.x

Drupal version: Drupal 6.x, Drupal 7.x
Audience: Themers
Page status: Needs updating
Last modified: October 7, 2011

NOTE: We are now using a different system to track version-to-version changes in Drupal. Most of the theme changes between Drupal 6 and Drupal 7 are noted on this page, but changes that were recorded after we adopted the new system can be found here:
http://drupal.org/list-changes/drupal
To add a new change notice:
http://drupal.org/node/add/changenotice
To find changes that need to be updated in the documentation:
http://drupal.org/list-change-updates/drupal

Help with upgrading the handbook to incorporate these changes. #740194: Update theming guide for Drupal 7

Once an update in this list is updated in the handbook, add [U], as a link to the handbook page, so others know the update is complete.

Overview of Drupal Theme changes in 7.x

1. [U] Blocks have new, more meaningful CSS IDs
2. [U] Primary and secondary links are now Main and Secondary menu
3. [U] Unrendered taxonomy links are no longer available as a separate variable in node.tpl.php files
4. [U] RDFa requires some changes at the beginning of html.tpl.php
5. [U] The clear-block class has been renamed to clearfix
6. [U] The box.tpl.php template has been removed
7. $help became a region
8. [U] Mission statement removed, 'highlighted' region suggested
9. [U] Footer message removed
10. [U] Content region is now mandatory, main page content became a block
11. [U] Second phase variable process functions
12. HTML classes generated through a variable
13. HTML attributes generated through a variable
14. Variable process functions can now be used for all theming hooks
15. All theme functions now take a single argument, $variables
16. [U] Function names must match theme name
17. [U] All CSS and JavaScript files must be specified in the theme's .info file
18. Renamed $block->content in block.tpl.php
19. Granular rendering in node and user templates
20. [U] Added jQuery UI (1.8) to core
21. Attached JS/CSS for elements
22. $closure becomes $page_bottom, new $page_top and hidden regions
23. $left and $right variables are now $sidebar_first and $sidebar_second; CSS IDs also changed
24. $picture changes to $user_picture, and the CSS class 'picture' to 'user-picture'
25. [U] New classes available to hide content in an accessible manner
26. JavaScript variable Drupal.jsEnabled has been removed
27. PHPTemplate suggestion wildcard
28. Include theme definition explicitly on element when using system_elements()
29. Added markup to make installation task progress perceivable with screen-reader and CSS disabled.
30. Added an invisible heading to theme_breadcrumb().
31. Changes to alt and title attribute for the RSS feed icon
32. Search box moved from theme layer to blocks
33. Changes to menu tree, link and tab rendering functions
34. theme_links() has a new parameter 'heading' for accessibility
35. theme_links() now has some specific versions for more targeted overrides
36. theme_get_setting() and THEME_settings() have been improved
37. Added a theme_form_required_marker() function
38. Added a theme_link() function

39. Skip to main content links in core themes
40. Alter hooks available to themes
41. System module stylesheets have been reorganized to separate behavior-supporting styles from presentational styles
42. New theme setting for displaying the Shortcut module "add to shortcuts" link
43. Specific template overrides of generic templates use a '--' delimiter instead of '-'
44. CSS files are sometimes loaded with @import, sometimes with LINK tags
45. [U] Browser-targeted CSS files can and should be added using drupal_add_css()
46. Targeted overrides (suggestions) available for theme_menu_link() and theme_menu_tree()
47. theme_submenu() was removed
48. New $title_prefix and $title_suffix template variables
49. theme_node_form() was removed
50. node_get_types() renamed to node_type_get_types()
51. Core themes now contain "package = Core" in their .info files
52. Heading elements added to search-result.tpl.php
53. The name attribute in a and map elements is invalid
54. PHPTemplate is now the default theme engine
55. Custom regions must be printed differently in page.tpl.php
56. [U] Thumbnail size has changed
57. $show_blocks theme variable has been removed
58. Regions are now rendered with their own TPL file: region.tpl.php
59. New html.tpl.php file
60. HTML rendering of form elements has different CSS classes
61. Local tasks and actions can now be altered
62. theme('node_submitted') and theme('comment_submitted') no longer exist

Blocks have new, more meaningful CSS IDs

Many of the CSS IDs for blocks defined by Drupal core have changed so that they more clearly indicate the purpose of the block:

Recent blog posts
Old CSS ID (Drupal 6): *block-blog-0*
New CSS ID (Drupal 7): *block-blog-recent*

Book navigation
Old CSS ID (Drupal 6): *block-book-0*
New CSS ID (Drupal 7): *block-book-navigation*

Recent comments
Old CSS ID (Drupal 6): *block-comment-0*
New CSS ID (Drupal 7): *block-comment-recent*

Active forum topics
Old CSS ID (Drupal 6): *block-forum-0*
New CSS ID (Drupal 7): *block-forum-active*

New forum topics
Old CSS ID (Drupal 6): *block-forum-1*
New CSS ID (Drupal 7): *block-forum-new*

Language switcher
Old CSS ID (Drupal 6): *block-locale-0*
New CSS ID (Drupal 7): *block-locale-language-switcher*

Syndicate
Old CSS ID (Drupal 6): *block-node-0*
New CSS ID (Drupal 7): *block-node-syndicate*

Most recent poll
Old CSS ID (Drupal 6): *block-poll-0*
New CSS ID (Drupal 7): *block-poll-recent*

Author information
Old CSS ID (Drupal 6): *block-profile-0*
New CSS ID (Drupal 7): *block-profile-author-information*

Search form
Old CSS ID (Drupal 6): *block-search-0*
New CSS ID (Drupal 7): *block-search-form*

Popular content
Old CSS ID (Drupal 6): *block-statistics-0*
New CSS ID (Drupal 7): *block-statistics-popular*

Powered by Drupal
Old CSS ID (Drupal 6): *block-system-0*
New CSS ID (Drupal 7): *block-system-powered-by*

User login
Old CSS ID (Drupal 6): *block-user-0*
New CSS ID (Drupal 7): *block-user-login*

Navigation
Old CSS ID (Drupal 6): *block-user-1*
New CSS ID (Drupal 7): *block-system-navigation*

Who's new
Old CSS ID (Drupal 6): *block-user-2*
New CSS ID (Drupal 7): *block-user-new*

Who's online
Old CSS ID (Drupal 6): *block-user-3*
New CSS ID (Drupal 7): *block-user-online*

For example, a Drupal 6 CSS style declaration such as:

```
/* Make the text in the user login block bigger. */
#block-user-0 {
  font-size: 1.5em;
}
```

should become (in Drupal 7):

```
/* Make the text in the user login block bigger. */
#block-user-login {
  font-size: 1.5em;
}
```

[Docs Updated]

Primary and secondary links are now Main and Secondary menu

Primary and Secondary links have been renamed to Main and Secondary menu. Themes which support these options will need to be updated to use the new variable names:

6.x: page.tpl.php

```
  <div id="menu">
    <?php if (isset($secondary_links)) { ?><?php print theme('links', $secondary_links, array('class' => 'links',
'id' => 'subnavlist')); ?><?php } ?>
    <?php if (isset($primary_links)) { ?><?php print theme('links', $primary_links, array('class' => 'links', 'id' =>
'navlist')) ?><?php } ?>
  </div>
```

7.x: page.tpl.php

```
  <div id="menu">
    <?php if (isset($secondary_menu)) { ?><?php print theme('links', $secondary_menu, array('class' =>
'links', 'id' => 'subnavlist')); ?><?php } ?>
    <?php if (isset($main_menu)) { ?><?php print theme('links', $main_menu, array('class' => 'links', 'id' =>
'navlist')) ?><?php } ?>
  </div>
```

You will also need to make the appropriate variable name changes if your theme's theme.info is defining features[]. Defining renamed or replaced features may cause all features to render as blank or empty arrays.

6.x: theme.info - features[]

```
  features[] = primary_links
  features[] = secondary_links
```

7.x: theme.info - features[]

```
  features[] = main_menu
  features[] = secondary_menu
```

Also, if your theme.info is defining **features[] = mission** please note that this feature has been removed and replaced with a variable named $mission which can be output in your page template.

[Docs Updated]

Unrendered taxonomy links are no longer available as a separate variable in node.tpl.php files

Previously, node.tpl.php templates could use the `$taxonomy` variable if they needed access to an array of unrendered taxonomy links associated with the current node.

In Drupal 7, this is no longer the case. Instead, Taxonomy terms are handled by the Field module, which means terms are accessed the same way as other fields of a given node.

When a node is prepared for output and passes through the Drupal theming layer, a `$content` variable is passed to node.tpl.php. Inside of $content are structured arrays that represent the various fields that make up the current node.

The field type for a taxonomy term is "term reference". A term reference is added to a content type the same way a new image or text field would be added.

Each field has a machine name that begins with `field_`. If a term reference field is added for the vocabulary "Animals", for example, you might choose to assign that field the machine name `field_animals`.

To output the terms for `field_animals`, it would then be passed through the render() function. For example:

```
<?php
render($content['field_animals']);
?>
```

[Docs Updated]

RDFa requires some changes at the beginning of html.tpl.php

Drupal 7 is able to output RDFa which requires the following changes in html.tpl.php.

- The DOCTYPE must be changed to XHTML+RDFa 1.0.
- The old `lang` attribute should be removed for compatibility with XHTML 1.1, only `xml:lang` should remain.
- The RDF namespace prefixes used throughout the HTML document need to be serialized in the `html` tag and are contained in the `$rdf_namespaces` variable.
- The GRDDL profile should be specified in the `<head>` tag.

6.x

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php print $language->language ?>" lang="<?
php print $language->language ?>" dir="<?php print $language->dir ?>">
  <head>
```

7.x

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<?php print $language->language ?>" dir="<?php
print $language->dir ?>"
  <?php print $rdf_namespaces ?>>
  <head profile="<?php print $grddl_profile ?>">
```

[Docs Updated]

The clear-block class has been renamed to clearfix

The "clear-block" class was a Drupalism for functionality that was better known by the CSS Community as the "clearfix" class. Also, using "block" in the old name was confusing to Drupal users more familiar with Drupal's *block system* than with the properties inside that CSS ruleset. New themers and designers to the Drupal community will no longer be confused by the old class name.

If you had a `<div class="clear-block">` in your D6 theme, simply change it to `<div class="clearfix">`.

[Docs Updated]

The box.tpl.php template has been removed

The amorphous box.tpl.php template has been removed. Those pieces of content that were using the box.tpl.php now have their own theme functions.

The search results listings are just themed with theme('search_results') instead of being wrapped in an additional box. The comment form box is themed with a theme_comment_form_box() theme function.

[Docs Updated]

$help became a region

(issue) In Drupal 7, a new region was added to the defaults (`left`, `right`, `content`, `header`, `footer`) called `help`. By default, the textual content of this region is the same as the `$help` variable was in page.tpl.php for Drupal 6.

Themes in Drupal 7 need to ensure that the `$help` variable is printed in page.tpl.php and, if the theme overrode the default regions, that the following line is added to its .info file:

```
regions[help] = Help
```

The help text is now surrounded by the block.tpl.php template's `<div>` tags and classes, so the CSS used to style the help likely needs changing.

[docs updated]

Mission statement removed, 'highlighted' region suggested

(issue) In Drupal 6, the page template received a special variable called $mission, which contained the mission statement setting of the website when on the front page. Drupal 6 themes also had an option on the

theme settings page to toggle this functionality. Drupal 7 removes the mission setting and the option toggle in favor of the more general custom block placement in regions.

Drupal 7 core themes now include a region named 'highlighted' which uses the same display as the mission statement area in Drupal 6. Whether this region has content now depends on administrators setting block placement, and is not limited to the front page only. Content in the highlighted region will be surrounded by the block.tpl.php template's `<div>` tags and classes, so the CSS used to style this area might need changing.

6.x

In .info:

```
features[] = mission
```

In page.tpl.php:

```
<?php print $mission; ?>
```

7.x

If you have defined custom regions in your .info file, you may add the highlighted region to the existing list of regions as shown below. If your theme does not define any regions, the highlighted region will be provided by core automatically, and you'll only need to ensure that you print it in page.tpl.php.

in .info:

```
regions[highlighted] = Highlighted
```

in page.tpl.php:

```
<?php print render($page['highlighted']); ?>
```

[Docs Updated].

Footer message removed

(issue) In Drupal 6, the page template received a special variable called $footer_message, which contained the footer message setting of the website. This was usually output before the footer region ($footer) by the template. Drupal 7 recognizes that the footer message was just a special type of user defined block. Those who had this setting in Drupal 6 will get a user defined block in the update, positioned in the $footer region.

To update your themes, just remove the $footer_message variable from them.

If you happened to output the $footer_message in your page template, but did not yet support the $footer region, now might be the time to start supporting this region. If you don't override any regions, the footer region will be predefined for you. If you do override regions, please output the $footer content in your page template and include the footer region in your .info file:

```
regions[footer] = Footer
```

Support for the footer region is just suggested but not required by Drupal. Those upgrading from Drupal 6 with a theme lacking support for the footer region will be able to reposition their block to another region.

Content region is now mandatory, main page content became a block

In Drupal up to version 6, the $content variable in page.tpl.php contained the main page content appended with the blocks positioned into the content region (if you had that region defined).

In Drupal 7, $content became a full region and is now mandatory in all themes. This new requirement was set up so that when enabling new themes, Drupal knows where to put the main page content by default.

In Drupal 6, it was only possible to put blocks after the main page content in this region. The only way to put blocks before the main page content was to define a specific region for that purpose. Drupal 7 now makes the main page content its own block. This makes it possible to put blocks before or after the main page content in the region without hacking in a new region.

If you relied on the fact that blocks were only put in the sidebars and therefore got their styling via just a `.block` class selector or something similar, now you might need to revisit your CSS files. Because the main page content is wrapped by markup from block.tpl.php, the `.block` selector will match that too, so you need to limit your blocks styling to certain regions by making the selectors more specific, such as `#left-sidebar .block`.

Second phase variable process functions

There are now two sets of variables process functions. The first are the existing "*pre*process" functions. The second are "process" functions which are run after preprocessors. All the various prefixes and suffixes apply to this second phase in the exact same way. This is useful when certain variables need to be worked on in two phases.

For example, adding classes into an array for the "preprocess" phase then flattening them into a string in the "process" phase so it's ready to print within a template. See next section.

[Docs Updated].

HTML classes generated through a variable

All templates can now print out `$classes` from a template to render dynamic classes built from variable process functions. The way to add these dynamic classes is by feeding the variable key labeled "classes_array" like so:

```
<?php
function mytheme_preprocess_node(&$vars) {
  // Add a striping class.
  $vars['classes_array'][] = 'node-' . $vars['zebra'];
}
?>
```

The default "template_process" (second phase processor) will take care of flattening the array into a flat string making it ready to print from your template. Dynamic classes are generated for common templates by default but due to the way it's set-up, any template can have a $classes variable.

Example:

```
<div class="<?php print $classes ?>">
  ...
</div>
```

HTML attributes generated through a variable

All templates can now print out `$attributes`, `$title_attributes`, and `$content_attributes` from a template to render dynamic attributes built from variable process functions. The RDF module and other modules add important information to these variables, so it is important for themes to ensure that these variables are printed correctly within all overridden template files. These three variables contain attributes for the overall element being rendered by the template, and its primary title and content elements, respectively. The way to add attributes to these variables is by feeding the variable keys labeled "attributes_array", "title_attributes_array", and "content_attributes_array" like so:

```
<?php
function mytheme_preprocess_node(&$vars) {
  // If the node was saved with a log message and the currently logged in user
  // has permission to view that message, add it as a title attribute (tooltip)
  // when displaying the node.
  if (!empty($vars['node']->log) && user_access('view revisions')) {
    $vars['attributes_array']['title'] = $vars['node']->log;
  }

  // Force the direction of node titles to be left to right, regardless of
  // language or any other settings.
  $vars['title_attributes_array']['dir'] = 'ltr';
}
?>
```

The default "template_process" (second phase processor) takes care of flattening the arrays into strings ready to print within the template file. The flattening process results in either empty strings (if no dynamic attributes were set) or strings that have a leading space followed by attribute names and values. Because of this, the attributes variables should be printed directly next to what precedes it in the template, with no leading space.

Example:

```
<div id="..." class="..."<?php print $attributes; ?>>
  <h2<?php print $title_attributes; ?>>...</h2>
  <div class="content"<?php print $content_attributes; ?>>...</div>
</div>
```

As is shown in the example, the convention is for the "id" and "class" attributes to be printed explicitly within the template, and for the attributes variables to be used for all other attributes. Therefore, to ensure that no attribute gets printed twice within the same element, the following rules should be followed:

- Preprocess functions within modules must not add "id" or "class" to the attributes arrays.
- Preprocess functions within themes may add "id" and/or "class" to the attributes arrays, but if they do so, the theme must also override the corresponding template file and ensure that the same attribute isn't being printed explicitly.
- Templates must not print any attribute other than "id" or "class" explicitly on any element for which an attributes variable is also being printed. Instead, the theme must use a preprocess function, as shown above.

Variable process functions can now be used for all theming hooks

(issue) In Drupal 6, preprocess functions could only be used for theming hooks rendered by templates. In Drupal 7, hook-specific preprocess and process functions can be used for all theming hooks, whether rendered by templates or functions. For example, a theme can make all menu links that start with "http:" or

"https:" (as opposed to ones that refer to an internal Drupal path) to open in a new browser tab:

```
<?php
function mytheme_preprocess_menu_link(&$variables) {
  if (
    substr($variables['element']['#href'], 0, 5) == 'http:' ||
    substr($variables['element']['#href'], 0, 6) == 'https:'
  ) {
    $variables['element']['#localized_options']['attributes']['target'] = '_blank';
  }
}
?>
```

Every preprocess function adds to the time it takes to theme the item, so especially for theme functions that get called a lot, keep an eye on performance when adding preprocess functions.

To minimize performance overhead, the non-hook-specific preprocess and process functions are called for templates only. See the API documentation for theme() for the full list of hook-specific and non-hook-specific preprocess and process functions.

All theme functions now take a single argument, $variables

(issue) In Drupal 6, theme functions registered their function arguments in hook_theme(). In Drupal 7, all theme functions take a single argument, $variables, an array of keyed variables, and register the expected keys within this array in hook_theme().

6.x

```
<?php
function drupal_common_theme() {
  return array(
    ...
    'table' => array(
      'arguments' => array('header' => NULL, 'rows' => NULL, 'attributes' => array(), 'caption' =>
NULL),
    ),
    ...
  );
}

function theme_table($header, $rows, $attributes = array(), $caption = NULL) {
  ...
}
?>
```

7.x

```
<?php
function drupal_common_theme() {
  return array(
    ...
    'table' => array(
      'variables' => array('header' => NULL, 'rows' => NULL, 'attributes' => array(), 'caption' => NULL,
'colgroups' => array(), 'sticky' => TRUE),
    ),
    ...
  );
}

function theme_table($variables) {
  $header = $variables['header'];
  $rows = $variables['rows'];
  $attributes = $variables['attributes'];
  $caption = $variables['caption'];
  $colgroups = $variables['colgroups'];
  $sticky = $variables['sticky'];
  ...
}
?>
```

Function names must match theme name

Function names in the file `template.php` must now use the relevant theme's name. You may no longer use phptemplate_function. This change was made in the following patch: Die, themeEngineName_ prefix, die!. To update your theme ensure there are no functions that begin with the template engine name (phptemplate) in the file `template.php` or any related template files.

All CSS and JavaScript files must be specified in the theme's .info file

In Drupal 6 `style.css` and `script.js` would be included in your theme automatically, even if they weren't present in the theme's .info file. In Drupal 7, themes must specify these files in the .info file to include them. More information about this change can be seen at #351487: Remove default values for stylesheet and scripts includes from system module. If your theme doesn't use these files, or if they are already specified in your theme's info file, no changes are required. *[docs updated]*

Renamed $block->content to $content in block.tpl.php

See this issue for the whole story.

Granular rendering in node and user templates.

Issue. Template authors may now finally print out bits of node and profile content as they see fit and still maintain forward compatibility with new modules that might add new content. To do so, template authors should use 2 new functions - render() and hide(). Example taken from node.tpl.php:

```
<div class="content">
 <?php
  // We hide the comments and links now so that we can render them later.
  hide($content['comments']);
  hide($content['links']);
  print render($content);
 ?>
</div>

<?php print render($content['links']); ?>

<?php print render($content['comments']); ?>
```

render() returns all the items that are in $content. So, `print render($content)` is equivalent to the D6 `print $content`. When a templater wants to print out part of the `$content` array, she may do so with something like `print render($content['links'])`. If the printing of links comes after the printing of all the rest of `$content`, then templater should call `hide($content['links'])` before calling `print render($content)`. Then, the links can be printed further down in the template with `print render($content['links'])`.

Added jQuery UI (1.8) to core

(issue) jQuery UI 1.8 was added to core. You can find the jQuery UI files in misc/ui and can add Javascript and CSS files from there to your pages with the regular drupal_add_js() and drupal_add_css() calls, no special function calls required. If you are migrating a theme which was previously dependent on the jquery_ui contributed module, see the module update guide on the topic, which provides examples. *[docs updated]*

Attached JavaScript and CSS for drupal_render

(issue) Individual elements now have the ability to define what JavaScript and cascading stylesheets are associated with them. This is stated in the #attached_js and #attached_css property.

Drupal 6.x:

```
<?php
function example_admin_settings() {
 // Add example.admin.css
 drupal_add_css(drupal_get_path('module', 'example') .'/example.admin.css');
 // Add some inline JavaScript
 drupal_add_js('alert("You are visiting the example form.");', 'inline');
 // Add a JavaScript setting.
 drupal_add_js(array('mymodule' => 'example'), 'setting');
 $form['example'] = array(
  '#type' => 'fieldset',
  '#title' => t('Example');
 );
 return $form;
}
?>
```

Drupal 7.x:

```
<?php
function example_admin_settings() {
 $form['#attached_css'] = array(
  // Add example.admin/css.
  drupal_get_path('module', 'example') . '/example.admin.css'
 ),
 $form['#attached_js'] = array(
  // Add some inline JavaScript.
  'alert("You are visiting the example form.");' => 'inline',
  // Add a JavaScript setting. Note that when the key is a number, the 'data' property will be used.
```

```
    array(
      'data' => array('mymodule' => array(...)),
      'type' => 'setting'
    ),
  );
  $form['example'] = array(
    '#type' => 'fieldset',
    '#title' => t('Example');
  );
  return $form;
}
?>
```

$closure becomes $page_bottom, new $page_top and hidden regions

(issue 1), (issue 2) Drupal 6 provides a special variable called $closure which should be put at the bottom of the HTML body output and can be themed via theme_footer() (which calls out to implementations of hook_footer() in modules). To generalize on one way to put output to the different page areas, Drupal 7 standardizes on regions and introduced the page_bottom region in place of the $closure special variable. Also, page_top is added as a pair of page_bottom. In Drupal 7 you need to output $page_top at the top of the HTML body output and $page_bottom at the bottom.

Drupal 6 (page.tpl.php):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
...
<body class="<?php print $body_classes; ?>">
...
    <?php print $closure; ?>
</body>
</html>
```

Drupal 7 (html.tpl.php):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
...
<body class="<?php print $classes; ?>">
...
  <?php print $page_top; ?>
  <?php print $page; ?>
  <?php print $page_bottom; ?>
</body>
</html>
```

If you define custom regions, it is important to remember that you need to include the page_top and page_bottom regions in your set of regions.

theme .info file extract:

```
regions[content] = Content
regions[help] = Help
regions[page_top] = Page top
regions[page_bottom] = Page bottom
```

The page_top and page_bottom regions are hidden, which means they will not show up on the blocks administration interface. When doing site-specific themes, it might also be useful to add more hidden regions (to provide ways for modules to add output to more places in the theme without defining blocks showing up on the blocks interface), you can do that via the regions_hidden[] .info file array which is new to Drupal 7:

theme .info file extract:

```
regions[content] = Content
regions[help] = Help
regions[page_top] = Page top
regions[page_bottom] = Page bottom
regions[indicators] = Indicators
regions_hidden[] = indicators
```

$left and $right variables are now $sidebar_first and $sidebar_second; CSS IDs also changed

(issue) In Drupal 6, the sidebar variable names were `$left` and `$right`. In Drupal 7 they are `$sidebar_first` and `$sidebar_second`.

6.x

```
    <?php if (!empty($left)): ?>
```

```
    <div id="sidebar-left" class="column sidebar">
      <?php print $left; ?>
    </div> <!-- /sidebar-left -->
  <?php endif; ?>
...
  <?php if (!empty($right)): ?>
    <div id="sidebar-right" class="column sidebar">
      <?php print $right; ?>
    </div> <!-- /sidebar-right -->
  <?php endif; ?>
```

7.x

```
  <?php if ($sidebar_first): ?>
    <div id="sidebar-first" class="column sidebar"><div class="section region">
      <?php print $sidebar_first; ?>
    </div></div> <!-- /.section, /#sidebar-first -->
  <?php endif; ?>

  <?php if ($sidebar_second): ?>
    <div id="sidebar-second" class="column sidebar"><div class="section region">
      <?php print $sidebar_second; ?>
    </div></div> <!-- /.section, /#sidebar-second -->
  <?php endif; ?>
```

This means that associated CSS IDs have changed as well:

Old CSS ID (Drupal 6)	New CSS ID (Drupal 7)
#sidebar-left	*#sidebar-first*
#sidebar-right	*#sidebar-second*

[docs updated]

$picture changes to $user_picture, and the CSS class 'picture' to 'user-picture'

The variable and the CSS class have been renamed to be more descriptive.

Drupal 6 (user-picture.tpl.php):

```
<div class="picture">
  <?php print $picture; ?>
</div>
```

Drupal 7:

```
<?php if ($user_picture): ?>
  <div class="user-picture">
    <?php print $user_picture; ?>
  </div>
<?php endif; ?>
```

New classes available to hide content in an accessible manner

Three new system classes have been added to be used for the purpose of hiding elements, `.element-hidden`, `.element-invisible`, and `.element-invisible.element-focusable`. Each class serves its own unique purpose:

`.element-hidden`
: The role of this class is to hide elements from all users. This class should be used for elements which should not be immediately displayed to any user. An example would be a collapsible fieldset that will be expanded with a click from a user. The effect of this class can be toggled with the jQuery `show()` and `hide()` functions.

`.element-invisible`
: The role of this class is to hide elements visually, but keep them available for screen-readers. This class should be used for information required for screen-reader users to understand and use the site where visual display is undesirable. Information provided in this manner should be kept concise, to avoid unnecessary burden on the user. This class must not be used for focusable elements (such as links and form elements) as this causes issues for keyboard only or voice recognition users.

`.element-invisible.element-focusable`
: The .element-focusable class extends the .element-invisible class to allow the element to be focusable when navigated to via the keyboard.

(Added to Standard Drupal core styles and classes.)

JavaScript variable Drupal.jsEnabled has been removed

In previous versions of Drupal, you could do the following in JavaScript code, to verify that JavaScript was enabled and sufficient for Drupal to do its "behaviors":

```
if( Drupal.jsEnabled ) {
  // something
}
```

In Drupal 7, the `Drupal.jsEnabled` variable is no longer defined, and there is no work-around -- the assumption is that JQuery things will simply not work if they don't work, so there's no reason to check ahead of time. See issue ~~#444352: Kill the killswitch~~ for discussion.

PHPTemplate suggestion wildcard

(issue) PHPTemplate offers suggestions based on the URI integers. In Drupal 6, you have to theme the previous suggestion or the specific one eg.

page-user.tpl.php or page-user-1.tpl.php

This was cumbersome because to theme all the user pages, it meant overriding page-user.tpl.php, which in turn themed the user login page.

The new suggestion wildcard for integer arguments accepts % suggestions page--user--%.tpl.php. Suggestions which have additional arguments like page--user--edit.tpl.php remain the same; this simply differentiates the suggestions with and without integer args.

Include theme definition explicitly on element when using system_elements()

In system_elements(), it is now necessary to include the theme definition explicitly on the element, rather than allowing the system to "fallback" and assign it automatically. Refer to issue 127731.

Added markup to make installation task progress perceivable with screen-reader and CSS disabled

(Issue)

In D6 there is no markup to indicate which of the installation tasks were complete, or which task is active, these differences are only shown with CSS styling.

In Drupal 7 markup has been added to theme_task_list() to:

Add a heading that is only visible to screen-reader users and when CSS is disabled.

```
<h2 class="element-invisible">Installation tasks</h2>
```

Append text "(done)" and "(active)" to relevant tasks in the task list, visible only to screen-reader users and when CSS is disabled.

```
<span class="element-invisible">(done)</span>
```

Add an invisible heading to theme_breadcrumb()

(Issue)

In D6 there is no markup to indicate the role or purpose of the breadcrumb links to users of screen-readers.

Drupal 7 provides a heading before the list of breadcrumb links, which is visible to screen-reader users and when CSS is disabled. This heading has been added to theme_breadcrumb() and to garland_breadcrumb().

```
<h2 class="element-invisible">You are here</h2>
```

Changes to alt and title attribute for the RSS feed icon

In D6 the alt attribute of the RSS feed icon themed by theme_feed_icon() was statically set to "Syndicate content" and the title attribute of the icon was set to the string passed to the function in the $title parameter.

Drupal 7 sets the alt attribute of the image and the title attribute of the anchor for the RSS feed to the same text. The text is comprised of "Subscribe to " + the $title string passed to theme_feed_icon().

Search box moved from theme layer to blocks

In previous versions, the search_box was displayed by the theme using $search_box.

Drupal 7 removes this. The search form is simply part of the block system. Theme developers will want to remove all references to $search_box in the theme, and may need to change CSS to handle the search box in the block.

Site maintainers upgrading sites from Drupal 6 to Drupal 7 will want to add the search form block to their site.

Changes to menu tree, link and tab rendering functions

(Issue)

function menu_tree_output() now returns a structured array of data that can be rendered to html using drupal_render().

Function theme_menu_item_link() and theme_menu_item() have been removed and are effectively replaced by in terms of rendering a menu tree by theme_menu_link(). This is the default function used to render each link in the tree returned by menu_tree_output().

Function theme_menu_local_task() takes different parameters and has a companion theme_menu_local_action() that did not exist in Drupal 6. Both of these function take as the first parameter

a menu link array with 'title', 'href', and 'localized_options' keys. In Drupal 6, the first parameter was a string. The change to array was made in order to allow removal of theme_menu_item_link(). The second parameter theme_menu_local_task() of is still a boolean, $active.

theme_links() has a new parameter 'heading' for accessibility

Issue: ~~#364219: Navigation menus should be preceded by headings of the appropriate level (usually <h2>).~~

To meet Web Content Accessibility Guidelines (WCAG) requirements, HTML headings should be used before all content sections, which includes lists of links such as menus. The headers ensure that there is a quick way for assistive technology users to browse through the content; however, most visual users do not need headers on navigation lists, because they can get a sense of the structure by how they are arranged visually on the page. So, the recommendation is to add a header with the "element-invisible" CSS class on it, so that only assistive technology users will notice the header.

It is also important that the header level used (H2, H3, etc.) be appropriately nested in the heading hierarchy. So, it is not recommended to just blindly add an H2 header before each list.

For that reason, the theme_links() function, which is normally called via theme('links', $variables), has a new third variable 'heading' to add the required heading to a list of links.

For example - Drupal 6 in a typical page.tpl.php file:

```
<?php print theme('links', $secondary_menu, array('id' => 'secondary-menu', 'class' => array('links', 'clearfix'))); ?>
```

Drupal 7:

```
<?php print theme('links', array('links' => $main_menu, 'attributes' => array('id' => 'main-menu', 'class' => array('links', 'clearfix')), 'header' => array('text' => t('Main menu'), 'level' => 'h2', 'class' => array('element-invisible')))); ?>
```

This will result in a semantically correct and accessible secondary menu in Drupal 7, because an invisible heading has been added:

```
<h2 class="element-invisible">Secondary menu</h2>
```

For more information:

- Accessibility in Drupal
- Theming accessibility guide
- Headings section of theme accessibility guide

theme_links() now allows more targeted overrides

(issue) Modules can now specify a more specific theme callback for links, so that theming of links can be overridden for specific purposes.

For instance, the Node module uses 'links__node' rather than just 'links' as its theme callback when adding links to nodes, so if you want to override just the theming of these links, you can do it in a mytheme_links__node() function, rather than adding some complicated logic to the the generic mytheme_links() function.

Other examples in Drupal core are links__contextual, links__system_main_menu, links__system_secondary_menu, etc.

theme_get_setting() and THEME_settings() have been improved

In Drupal 6, themes could add custom form elements to their "configure theme settings" page at admin/build/themes/settings/THEMENAME. Themes would need to create a theme-settings.php page in their theme directory and use a function with the following syntax:

```
<?php
/**
 * 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) { }
?>
```

In Drupal 7, much more flexibility is given to themes to modify the entire theme settings form. In a theme's theme-settings.php, themes should now use a THEMENAME_form_system_theme_settings_alter(&$form, $form_state) function. This gives the same power to themes that modules have if they use hook_form_system_theme_settings_alter(). See the "Forms API Quickstart Guide" and "Forms API Reference" on http://api.drupal.org/api/7, as well as the hook_form_FORM_ID_alter() docs to learn the full flexibility of Forms API. Note that themes can no longer use the phptemplate_ prefix to the function; you'll need to use the actual name of your theme as the prefix.

Here's an example if you had a "foo" theme and wanted to add a textfield whose default value was "blue bikeshed":

```
<?php
function foo_form_system_theme_settings_alter(&$form, $form_state) {
  $form['foo_example'] = array(
    '#type'          => 'textfield',
    '#title'         => t('Widget'),
    '#default_value' => theme_get_setting('foo_example'),
    '#description'   => t("Place this text in the widget spot on your site."),
  );
}
?>
```

In order to set the default value for any form element you add, you'll need to add a simple line to your .info file: `settings[SETTING_NAME] = DEFAULT_VALUE`. For our foo theme, you'd need to edit the foo.info file and this line:

```
settings[foo_example] = blue bikeshed
```

In any of your theme's php files, you can retrieve the value the user set by calling:

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

theme_form_required_marker()

(issue) The markup for generating the marker on required form elements is now handled by a separate function, `theme_form_required_marker()` instead of just being included as part of the work performed by `theme_form_element()`. This allows you to reuse this markup in other places or to modify the markup without changing all of `theme_form_element()`

theme_link()

(issue) The markup for generating a link is now handled by a theme function, theme_link(), instead of being hard-coded into the `l()` function. This allows you to implement a preprocess function or an override function in order to customize the markup of any link. Doing so may slow down Drupal pages that have many links, so it is recommended to evaluate the benefit vs. the performance trade-off, but the capability exists for sites and themes that need it. This theme function is for customizing the markup of links, generically. Other theme functions exist for customizing links based on context. For example, to customize menu links, override `theme_menu_link()` instead.

Example of hook_preprocess_link():

```
<?php
function mytheme_preprocess_link(&$variables) {
  // In order to style links differently depending on what they are linking to,
  // add classes that contain information about the link path.
  if (strpos($variables['path'], ':') !== FALSE) {
    // For external links, add a class indicating an external link and a class
    // indicating the scheme (e.g., for 'mailto:...' links, add a 'link-mailto'
    // class).
    $variables['options']['attributes']['class'][] = 'link-external';
    $variables['options']['attributes']['class'][] = 'link-' . parse_url($variables['path'], PHP_URL_SCHEME);
  }
  else {
    // For internal paths, add a class indicating an internal link.
    $variables['options']['attributes']['class'][] = 'link-internal';
    if (empty($variables['path']) || $variables['path'] == '<front>') {
      // Add a class indicating a link to the front page.
      $variables['options']['attributes']['class'][] = 'link-front';
    }
    else {
      // For internal links not to the front page, add a class for each part
      // of the path. For example, for a link to 'admin/structure/block', add
      // the classes 'link-path-admin', 'link-path-admin-structure', and
      // 'link-path-admin-structure-block'.
      $class = 'link-path';
      foreach (explode('/', $variables['path']) as $path_part) {
        $class .= '-' . $path_part;
        $variables['options']['attributes']['class'][] = $class;
      }
    }
  }
}
?>
```

Example of overriding theme_link():

```
<?php
function mytheme_link($variables) {
  // Place a span within and outside the anchor tag in order to implement some
  // special styling.
  return '<span class="link-wrapper"><a href="' . check_plain(url($path, $options)) . '" .
drupal_attributes($options['attributes']) . '><span class="link-content-wrapper">' . ($options['html'] ?
$text : check_plain($text)) . '</span></a></span>';
}
?>
```

Skip to main content links in core themes

(Issue)
To meet Web Content Accessibility Guidelines (WCAG) 2.0 guideline 2.4.1 Bypass Blocks, web pages need to have a link to help keyboard only users and screen readers more easily access the main content of a website.

Garland's implementation is hidden visually, but keep them available for screen-readers. Furthermore, if a keyboard only user tabs through a site the link will become visible as it gains focus.

To hide the skip navigation you can use one the new classes available to hide content in an accessible manner.

Alter hooks available to themes

(Issue) Hooks that are used to alter content before being displayed on the page are now available to themes. Some important ones to note are:

- hook_page_alter
- hook_form_alter
- hook_js_alter
- hook_css_alter

Note that although technically all of the alter hooks are exposed to the theme, only a given number of them will actually work due to the way the Drupal bootstrap works. If you need to use hook_menu_alter, for example, you'll have to use a module. These hooks can be exposed in template.php.

Drupal 7.x:

```
<?php
/**
 * Implement hook_page_alter().
 */
function mytheme_page_alter(&$page) {
  // Remove elements from the page.
}

/**
 * Implement hook_css_alter().
 */
function mytheme_css_alter(&$css) {
  // Replace some CSS files with this theme's special CSS.
}
?>
```

System module stylesheets have been reorganized to separate behavior-supporting styles from presentational styles.

(Issue) Summary of changes:

- Styles from **default.css** were merged into **system.css** and **default.css** was removed.
- **system-behavior.css** was added to house behavior-supporting styles, ie. styles for autocomplete, drag and drop, collapsible fieldsets, progress bars, etc.

New theme setting for displaying the Shortcut module "add to shortcuts" link

Issue: ~~#674394: The add/remove shortcut buttons look bad and don't appear in most themes besides Seven~~

In Drupal 7, you will notice that when the Shortcut module is enabled, the core Seven administration theme displays a "plus" or "minus" sign next to the title of each page (for sufficiently privileged users), which allows that page to be added or removed from the user's set of shortcuts via a one-click link.

The appearance of this link is controlled by a theme setting. If you want it to display in your theme when the Shortcut module is enabled, add the following line of code to your theme's .info file:

```
settings[shortcut_module_link] = 1
```

Specific template overrides of generic templates use a '--' delimiter instead of

Issue ~~#678714: Unify use of theme hook / template suggestions, fix clobbering problems, and improve suggestion discovery performance~~

In Drupal 6, some template files could be overridden in a targeted way. For example, the theme could contain a "node-article.tpl.php" file which would be used for nodes of the article type only. In Drupal 7, these need to be renamed to use a double-hyphen. For example, "node--article.tpl.php". A single hyphen is still used to separate words: for example, "user-picture.tpl.php" or "node--long-content-type-name.tpl.php", so the double hyphen always indicates a more targeted override of what comes before the "--". As another example, in Drupal 7 the override of a page.tpl.php for node/17 is page--node--17.tpl.php (in Drupal 6 it would have been page-node-17.tpl.php).

Please also see http://drupal.org/update/modules/6/7#theme_hook_suggestions_2 for how this affects preprocess functions if your theme implements those.

CSS files are sometimes loaded with @import, sometimes with LINK tags

(Issue) Prior to Drupal 6, CSS files were added to the page using @import statements inside of STYLE tags. Drupal 6 switched to using LINK tags. With Drupal 7, LINK tags are used when the performance setting to "Aggregate and compress CSS files into one file" is enabled, but @import statements are sometimes used when that setting is disabled. This was needed to work around an Internet Explorer limitation of only being able to load the first 31 tags that add CSS files. See the API documentation for drupal_pre_render_styles() for more details explaining when LINK tags are used and when @import statements are used.

Browser-targeted CSS files can and should be added using drupal_add_css()

(Issue) Drupal 7 adds the ability to specify a 'browsers' key when calling drupal_add_css().

Drupal 6:
Garland's page.tpl.php:

```
<?php
...
<?php print $styles ? >
<!--[if lt IE 7]>
  <?php print phptemplate_get_ie_styles(); ? >
<![endif]-->
...
?>
```

Garland's template.php:

```
<?php
...
function phptemplate_get_ie_styles() {
  global $language;

  $iecss = '<link type="text/css" rel="stylesheet" media="all" href="'. base_path() . path_to_theme()
.'/fix-ie.css" />';
  if ($language->direction == LANGUAGE_RTL) {
    $iecss .= '<style type="text/css" media="all">@import "'. base_path() . path_to_theme() .'/fix-ie-
rtl.css";</style>';
  }

  return $iecss;
}
...
?>
```

Drupal 7:
Garland's template.php:

```
<?php
...
function garland_preprocess_html(&$vars) {
  ...
  drupal_add_css(path_to_theme() . '/fix-ie.css', array('group' => CSS_THEME, 'browsers' => array('IE'
=> 'lt IE 7', '!IE' => FALSE), 'preprocess' => FALSE));
}
...
?>
```

See the API documentation for drupal_pre_render_conditional_comments() for details on the 'IE' and '!IE' keys.

It is recommended for themes to always use drupal_add_css() for adding a CSS file, so that Drupal code knows the exact total number of CSS files being added, information that might be needed for working around Internet Explorer's limitation of only being able to load the first 31 LINK/STYLE tags. *[docs updated]*

Targeted overrides (suggestions) available for theme_menu_link() and theme_menu_tree()

(issue) In addition to other changes to menu rendering, a Drupal 7 theme can implement a `THEMENAME_menu_tree__MENU_NAME()` and/or `THEMENAME_menu_link__MENU_NAME()` function to override theme_menu_tree()/theme_menu_link() for a specific menu. For example, `THEMENAME_menu_link__management()` would override theme_menu_link() for links within the "Management" menu. This is similar to how "node--article.tpl.php" overrides "node.tpl.php".

theme_submenu() was removed

(issue) Drupal core does not provide theme_submenu() anymore.

New $title_prefix and $title_suffix template variables

(issue) Templates in Drupal 7 have two new standard variables, $title_prefix and $title_suffix, which are renderable arrays that contain output intended to be displayed before or after (respectively) the main title tag that appears in the template.

All standard templates that potentially have a title (e.g., nodes, blocks, comments, the main page.tpl.php file, etc.) should render these variables. It is important that the variables be rendered even if the title itself is not being displayed, since the variables might contain important data added by modules (for example, contextual links) associated with the template as a whole.

Example (from page.tpl.php):

```
<?php print render($title_prefix); ?>
<?php if ($title): ?><h1 class="title" id="page-title"><?php print $title; ?></h1><?php endif; ?>
<?php print render($title_suffix); ?>
```

theme_node_form() was removed

(issue) Drupal core does not provide theme_node_form() anymore. Node forms now have a CSS classes of .node-form and .node-TYPE-form for facilicate simple node type specific styling of the node form.

node_get_types() renamed to node_type_get_types()

In template.php, replace:

```
foreach (node_get_types() as $type => $name) {
unset($settings['toggle_node_info_' . $type]);
}
```

with:

```
foreach(node_type_get_types() as $type => $name) {
unset($settings['toggle_node_info_'. $type]);
}
```

Core themes now contain "package = Core" in their .info files

(issue) Each core theme now contains the line: `package = Core` in their .info files. This is a **core only** property that was added to to help the Update Manager module identify core modules and themes. It should not be used in custom or contributed themes.

search-result.tpl.php now uses proper H3 headings for search result titles

(issue) H3 heading elements are now used to wrap each search result title to allow for heading based navigation for keyboard only users (accessibility improvement). This required a change from a DL list to an OL list with changes in two templates - search-results.tpl.php and seach-result.tpl.php.

(second issue) Rendering was pushed to the last possible moment in search results theming, resulting in more changes to these templates and the pre-processing functions. One note: the $type variable is now $module (this variable is not used in the default search result markup).

Drupal 6 markup for search-results.tpl.php:

```
<dl class="search-results <?php print $type; ?>-results">
  <?php print $search_results; ?>
</dl>
<?php print $pager; ?>
```

Drupal 7 markup for search-results.tpl.php:

```
<?php if ($search_results) : ?>
  <h2><?php print t('Search results');?></h2>
  <ol class="search-results <?php print $module; ?>-results">
    <?php print $search_results; ?>
  </ol>
  <?php print $pager; ?>
```

```
<?php else : ?>
  <h2><?php print t('Your search yielded no results');?></h2>
  <?php print search_help('search#noresults', drupal_help_arg()); ?>
<?php endif; ?>
```

Drupal 6 markup for search-result.tpl.php:

```
<dt class="title">
  <a href="<?php print $url; ?>"><?php print $title; ?></a>
</dt>
<dd>
  <?php if ($snippet) : ?>
    <p class="search-snippet"><?php print $snippet; ?></p>
  <?php endif; ?>
  <?php if ($info) : ?>
  <p class="search-info"><?php print $info; ?></p>
  <?php endif; ?>
</dd>
```

Drupal 7 markup for search-result.tpl.php:

```
<li class="<?php print $classes; ?>"<?php print $attributes; ?>>
  <?php print render($title_prefix); ?>
  <h3 class="title"<?php print $title_attributes; ?>>
    <a href="<?php print $url; ?>"><?php print $title; ?></a>
  </h3>
  <?php print render($title_suffix); ?>
  <div class="search-snippet-info">
    <?php if ($snippet) : ?>
      <p class="search-snippet"<?php print $content_attributes; ?>><?php print $snippet; ?></p>
    <?php endif; ?>
    <?php if ($info) : ?>
      <p class="search-info"><?php print $info; ?></p>
    <?php endif; ?>
  </div>
</li>
```

The corresponding pre-processing functions for search results have also changed.

The name attribute in a and map elements is invalid

(issue) Due to Drupal 7's XHTML+RDFa 1.0 doctype (which inherits the HTML 1.1 doctype), the HTML output should be compatible with XHTML 1.1, in particular there should be no `name` attribute in the `a` and `map` HTML elements.

6.x

```
<a id="new" name="new">
```

7.x

```
<a id="new">
```

PHPTemplate is now the default theme engine

In Drupal 6, specifying the theme engine in .info files was required:
`engine = phptemplate`

In Drupal 7, this line is no longer necessary because it is assumed by default. You may safely remove it from your .info file.

Themes using the Smarty engine are not affected, and PHP only themes may still be used by specifying "theme" as the engine:
`engine = theme`

Custom regions must be printed differently in page.tpl.php

In Drupal 6, when you wanted to print a region you had just made, you just put this into your page.tpl.php:

```
<?php
print $foo_sidebar;
?>
```

In Drupal 7, all of the regions you wish to add must be printed using render and the page variable:

```
<?php
print render($page['foo_sidebar']);
?>
```

Thumbnail size has changed

(issue) The Appearance page in Drupal 7 (Themes in Drupal 6) was substantially reorganized. Accordingly, the size of the thumbnail to include with a theme in Drupal 7 changed. In Drupal 7, the thumbnail file should be 294px wide by 219px tall. See http://drupal.org/node/647754 for instructions on how to make your thumbnail file.

$show_blocks theme variable has been removed

(issue) The $show_blocks theme variable for pages, which used to be FALSE on pages such as 404 errors to indicate that the sidebars should not be shown, has been removed in Drupal 7. The reasoning was that it wasn't really possible for the theme system to know, for a given theme, which regions were "non-essential sidebars" and shouldn't be shown, and which were essential regions (navigation etc.).

In addition, theme('maintenance_page') no longer has a $show_blocks argument.

Regions are now rendered with their own TPL file: region.tpl.php

(issue) All theme regions, when rendered in the page, are rendered using a new region.tpl.php file. This ensures that regions are rendered consistently, and the template_preprocess_region() function ensures that each region has a consistent class attribute applied to it (classes: region, region-REGIONNAME).

Example - Drupal 6 - page.tpl.php

```
<div id="header-region" class="clear-block"><?php print $header; ?></div>
```

Drupal 7:

```
    <?php print render($page['header']); ?>
```

New html.tpl.php file

html.tpl.php takes over part of the Drupal 6 page.tpl.php, by providing the structure for the HTML header (the `<!DOCTYPE>` and the `<html>`, `<head>`, and `<body>` elements. See the documentation for html.tpl.php at api.drupal.org and the source code for html.tpl.php for the default html.tpl.php file, which of course can be overridden in a theme, just like any other tpl.php file.

HTML rendering of form elements has different CSS classes

Issue: The HTML rendering of form elements has changed, in that they have different CSS classes applied to the surrounding DIV: .form-type-TYPE and .form-item-NAME. Previously, the assigned classes were .form-item-TYPE and .NAME-wrapper. In both cases, TYPE is the #type component of the form element, and NAME is the array key. For example, in:

```
$form['body'] = array(
  '#type' => 'textarea',
  ...
);
```

TYPE would be textarea, and NAME would be body. (For multi-word types and names, underscores are converted to hyphens.)

Local tasks and actions can now be altered

(issue).
Previously, theme_menu_local_tasks() took no $variables, so preprocess functions on it were impossible. Modules can now alter the primary and secondary tabs using preprocess or process functions.

Themes that overrode the old definition of theme_menu_local_tasks() will need to update their theme declaration to mirror the changes to the default implementation in includes/menu.inc.

theme('node_submitted') and theme('comment_submitted') no longer exist

(issue). In Drupal 6, the template_preprocess_node() and template_preprocess_comment() functions called theme('node_submitted') and theme('comment_submitted') respectively, in order to create the "Submitted by" lines, which come into the node.tpl.php and comment.tpl.php files as variable $submitted. In Drupal 7, these two theme calls have been eliminated, and now the submitted information is generated directly in template_preprocess_node() and template_preprocess_comment().

So, if you want to override how the submitted by information is presented in a theme, you have two choices:

1. In your theme's mytheme_preprocess_node()/comment() function, do something different in place of:
   ```
   $variables['submitted'] = t('Submitted by !username on !datetime', array('!username' =>
   $variables['name'], '!datetime' => $variables['date']));
   ```
2. Override node.tpl.php/comment.tpl.php and use the $name and $date variables to do something different in the section:
   ```
   <?php if ($display_submitted): ?>
     <div class="submitted">
       <?php print $submitted; ?>
     </div>
   ```

```
<?php endif; ?>
```

‹ Updating a theme to a new version up Converting 5.x themes to 6.x ›

Comments

Should <?php if ($taxonomy):

Posted by redndahead on *June 23, 2009 at 5:08am*

Should

```
<?php if ($taxonomy): ?>
    <div class="terms"><?php print $terms ?></div>
  <?php endif;?>
```

Be

```
<?php if ($taxonomy): ?>
    <div class="terms"><?php print $taxonomy ?></div>
  <?php endif;?>
```

For D6?

I haven't used $taxonomy before so I'm not sure.

Should be $terms

Posted by NotUnderMe2 on *July 4, 2009 at 6:28pm*

Use $terms because is available in 6 and 7

$terms in d7?

Posted by adub on *January 2, 2011 at 9:09pm*

in d7 $terms didn't work for me but this did:

```
<?php print render($content['field_tags']); ?>
```


Drupal 7 Print Tags not working

Posted by dale brendan on *January 19, 2011 at 3:38pm*

Can not seem to get any of these to print in node.tpl.php for Drupal 7

node_get_types() no longer

Posted by Jacine on *August 5, 2009 at 1:58am*

node_get_types() no longer exists. This only affects themes that have theme settings. The function is called when initializing theme settings in template.php:

6.x

```
<?php
foreach (node_get_types() as $type => $name) {
  unset($settings['toggle_node_info_'. $type]);
}
?>
```

7.x

```
<?php
foreach (node_type_get_type() as $type => $name) {
  unset($settings['toggle_node_info_'. $type]);
}
?>
```

Login or register to post comments

Added note on node_type_get_types()

Posted by peterx on *April 26, 2010 at 8:26am*

Added note on node_type_get_types() after running into the problem in a conversion.

petermoulding.com/web_architect

Login or register to post comments

Improvements to drupal_add_js/css

Posted by Zarabadoo on *August 13, 2009 at 9:25pm*

It would be good to add information about the improvements about the new support for weights in `drupal_add_js()` and `drupal_add_css()`. The ability for inline styles via `drupal_add_css()` is also a good mention.

--
Al "Zarabadoo" Steffen

Login or register to post comments

drupal_add_css() in template.php

Posted by sociotech on *March 28, 2010 at 11:03pm*

Unless I'm missing something, a pretty big change that hasn't really been mentioned is that you can now use drupal_add_css() in a template.php preprocess function without having to also re-aggregate the stylesheets and then overwrite the $styles variable for page.tpl.php.

In D6 the $styles variable is already created from the css array by the time template.php runs, so adding css with drupal_add_css() has no effect unless you also re-create the $styles variable. And you have to take aggregation into account as well.

But in D7 it looks like the fix for adding conditional stylesheets (http://drupal.org/update/theme/6/7#browser-targeted-css) also implies that you can now generally use drupal_add_css() in a preprocess_html() function in template.php and have your changes correctly reflected in the $styles variable (aggregated or unaggregated) in page.tpl.php without having to do anything else.

I'm assuming that the same thing applies for drupal_add_js().

TopNotchThemes | Fusion Drupal Themes

Login or register to post comments

theme_blocks() does not exist anymore

Posted by JurriaanRoelofs on *September 13, 2010 at 7:09am*

If you used this for adding first/last classes to blocks in D6 you have to do this in the block preprocess file:

```
<?php
$block = $vars['block'];

// Special classes for blocks
$block_classes[] = 'block-'. $block->module;
$vars['classes_array'][] = 'region-'. $vars['block_zebra'];
$vars['classes_array'][] = $vars['zebra'];
$vars['classes_array'][] = 'region-count-'. $vars['block_id'];
$vars['classes_array'][] = 'count-'. $vars['id'];
$vars['classes_array'][] = ($vars['block_id'] == 1) ? 'first' : '';
$vars['classes_array'][] = ($vars['block_id'] == count(block_list($block->region))) ? 'last' : '';
?>
```

---
also known as peach
Still running SooperThemes Premium Drupal Themes

Login or register to post comments

RE: "PHPTemplate is now the

Posted by bradezone on *November 15, 2010 at 10:53pm*

RE: "PHPTemplate is now the default theme engine"
I've noticed that if you omit the "engine = phptemplate" line in a Drupal 7 theme, then the theme will not get refreshed when you clear your cache in performance settings. Until that bug is fixed, we may want to say that this line is still required.

Bradezone :: for all your Brade needs

Login or register to post comments

I also noticed this. If you

Posted by acke on *November 27, 2010 at 2:27pm*

I also noticed this. If you don't specifiy the theme engine in your themes .info file, Drupal 7 won't use your page.tpl.php but will use everything else. So I would also say this line is indeed required.

Login or register to post comments

Help

Posted by JRANGER on *July 6, 2011 at 2:24am*

Ok i am struggling with converting my drupal 6 theme to 7 due to this...

My drupal 6 is as follows
function phptemplate_node_submitted($node) {
return t('@day
@datetime',
array(
// '!username' => theme('username', $node),
'@day' => format_date($node->created,'custom','D'),
'@datetime' => format_date($node->created,'custom','m/d/y'),
));
}

function phptemplate_comment_submitted($comment) {
return t('@day @datetime by !username',
array(
'!username' => theme('username', $comment),
'@day' => format_date($comment->timestamp,'custom','D'),
'@datetime' => format_date($comment->timestamp,'custom','m/d/y'),
));
}
now i have replaced the phptemplate and tried various $vars['submitted'] =t('@day
@datetime',

but no matter what i do nothing changes? am i missing something?

Node is using just submitted and not the date stamps..ugh this is frustrating and im not a great php guy

Login or register to post comments

help

Posted by JRANGER on *July 6, 2011 at 2:25am*

Ok i am struggling with converting my drupal 6 theme to 7 due to this...

My drupal 6 is as follows
function phptemplate_node_submitted($node) {
return t('@day
@datetime',
array(
// '!username' => theme('username', $node),
'@day' => format_date($node->created,'custom','D'),
'@datetime' => format_date($node->created,'custom','m/d/y'),
));
}

function phptemplate_comment_submitted($comment) {
return t('@day @datetime by !username',
array(
'!username' => theme('username', $comment),
'@day' => format_date($comment->timestamp,'custom','D'),
'@datetime' => format_date($comment->timestamp,'custom','m/d/y'),
));
}
now i have replaced the phptemplate and tried various $vars['submitted'] =t('@day
@datetime',

but no matter what i do nothing changes? am i missing something?

Node is using just submitted and not the date stamps..ugh this is frustrating and im not a great php guy

Login or register to post comments

Script

Posted by saadmanna on *July 25, 2011 at 7:28pm*

Is there any PHP script to Convert Drupal6 theme to Drupal7.

Login or register to post comments

theme-link()

Posted by MainMain on *July 28, 2011 at 9:06am*

Hello,
I'm working with Drupal 7 and I had to build a drop down menu, I had also to add specific class to my links. I used menu block + theme_menu_link() in my template.php (I'm using zen, so I add the function : zen_menu_link($variables).

As it is said, to add a class, I had to do :

$variables['options']['attributes']['class'][] = 'link-front';

But It's not working for me. By printing $variables, I noticed that $variables is not exactly builded as announced. To customise class, I had to do :
$variables['element']['#attributes']['class'][] = 'link-front';

To print my menu, I had to do :
return theme_menu_link($variables);

And it is working fine.

Bye

Login or register to post comments