PLONE THEMING

Six Feet Up Quick Reference

DTML

CSS files linked with `base_properties.props` will allow you to use Document Template Markup Language (DTML) variables in your CSS.

The variables in `base_properties` will look like this:

```
fontColor:string=#666666
customVariable:string=Arial, sans-serif
```

To use these variables in your CSS, end your CSS filename with .dmtl: `main.css.dtml`. Do not include this extra extension when adding the file to the CSS Registry. Wrap your CSS code with these comments:

```
/* <dtml-with base_properties>
   <dtml-call "REQUEST.set('portal_url',
               portal_url())"> */

/* </dtml-with> */
```

Use the variables in your CSS with this syntax:
`&dtml-variableName;`

```
body {
  background: url('&dtml-portal_url;/bg.jpg');
  color: &dtml-fontColor;;
  font-family: &dtml-customVariable;;
}
```

Common Theming Stumbling Blocks

It is very common to make changes to your theme, then have the changes not display in the site. Here are the places to check to make sure everything is set up correctly for development:

theme installation
Check `portal_quickinstaller` to make sure your theme is installed.

debug mode
For CSS or JS files, make sure debug mode is checked in `portal_css` or `portal_javascripts`.

custom folder
If changes to a template on the filesystem aren't showing up, check `portal_skins > custom`. If the template also exists there, it's going to override your filesystem template.

thrive

Common Theming Stumbling Blocks *(continued)*

skins order
If an override isn't working, it could be the original product is being read in after your theme. Check `portal_skins > properties` tab to see the order of precedence. The top of the list should have custom, your theme, then everything else. You can reorder the layers here, but they may get reset when installing another product or theme. Change the order of installation so your theme's profile is run last.

xml files
When making a change to an xml file in the `profiles/default` folder of your theme, don't forget to import these changes in `portal_setup`.

Resources

Plone 3 Theming by Veda Williams (Packt Publishing)

Generic Setup Quick Reference Card
http://www.sixfeetup.com/swag

TAL Reference:
http://www.owlfish.com/software/simpleTAL/tal-guide.html

ZPT Style Guide:
http://wiki.zope.org/zope2/PageTemplateStyleGuide

ZPT docs from the Zope book:
http://docs.zope.org/zope2/zope2book/ZPT.html

Advanced ZPT docs from the Zope book:
http://docs.zope.org/zope2/zope2book/AdvZPT.html

A command line program to validate page templates:
http://pypi.python.org/pypi/zptlint

DTML Docs from the Zope book:
http://docs.zope.org/zope2/zope2book/DTML.html

This card contains handy tips and references to use in theming your Plone 3 or Plone 4 site.

Create a theme in your buildout with the command:

```
$ paster create -t plone3_theme
```

Helpful Product to Have In Your Buildout

collective.recipe.omelette
Allows easy browsing of the eggs used in your site.

z3c.jbot
Provides an easy way to override templates, without editing zcml for each template.

plone.reload
Reloads zcml and python code without restarting your instance.

Products.FSDump
Conveniently downloads everything from your custom folder.

Export Site Customizations to Your Theme

It's best to keep site customization within your theme in case you want to use the theme and settings elsewhere. Customizations can be exported from `ZMI > portal_setup`, and put into your `theme/profiles/default folder`. You don't need to keep everything in the exported xml file, only the customizations you made while keeping the proper xml structure intact. Here are some of the common options:

ZMI/site location : Title of export step : XML file
portal_actions : Action Providers : actions.xml
portal_css : Stylesheet Registry : cssregistry.xml
portal_javascripts : Javascript Registry : jsregistry.xml
@@manage-viewlets : Viewlet Settings : viewlets.xml

Adding New CSS and JS Files

New CSS and javascript files can be put into the skins path of your theme. The files will then need to be added to the appropriate registry, `portal_css` or `portal_javascripts`. The new entry can be added through the ZMI, or into your theme: `profiles/default/cssregistry.xml` or `profiles/default/jsregistry.xml`. If entered in the ZMI, you will need to export the registry via Generic Setup (`portal_setup`) to put the changes into your theme. Likewise, if the XML file was changed, you will need to import the registry into your site to see the changes.

These registries in Plone manage merging and caching of the CSS and JS files, to help with site performance.

Overriding Templates

A big part of Plone theming is knowing where to find the original files and knowing how to override those files properly. For files found in a skins path, creating a template or CSS file with the same name as an existing file in your theme's skin path will completely override the original. For CSS files, don't create an override if you are only making minor changes. Instead, put your custom CSS in `ploneCustom.css` or another file you create.

Where to Find Common Plone Templates

If you have the omelette set up, you will be able to find these files in `buildout/parts/omelette/`. Do not edit the templates here, but create an override in your theme.

- Templates for default content types: `Products/CMFPlone/skins/plone_content`
- Viewlet templates: `plone/app/layout/viewlets`
- Portlet templates: `plone/app/portlets/portlets`

Creating the Overrides

If the original template is found in a skins folder, you can override it by putting a copy in your theme's skins folder. For viewlets, portlets, or other files found in a browser folder, override using z3c.jbot.

Set Up z3c.jbot

Put the following code in `theme/browser/configure.zcml`:

```
<include package="z3c.jbot" file="meta.zcml" />
<browser:jbot
    directory="template-overrides"
    layer=".interfaces.IThemeSpecific" />
```

The layer may be different depending on your theme, but this will keep overrides specific to your theme.

Create a directory called `template-overrides` for your z3c.jbot overrides. Name the override by referencing the dotted path to the file: `plone.app.layout.viewlets.footer.pt`.

Styling Tips

For fixed width sites, set the width on #visual-portal-wrapper, not the body.

Try not to hide Plone elements with CSS. There is usually a better way that will prevent the elements from being rendered at all.

Check out the body tag for classes that will help you target certain items

- **template-folder_listing** - allows you to target items on the folder_listing template. Also tells you what template is in use, if you need to find it for overriding
- **section-news** - each section at the root of the site has its own class. Allows you to style different sections with different colors
- **portaltype-folder** - Allows you to apply type-specific styles.

TAL

The Template Attribute Language (TAL) is the language used in Zope Page Templates (ZPT) that allows your HTML to work with dynamic Plone content. Statements use TALES Expressions for presenting the content. There are three types of expressions:

Path Expressions
Path expressions simply evaluate the path given. Path expressions are the default and do not require a prefix like Python and string expressions do.

```
<tal:block define="my_var context/getMyVar" />
```

Python Expressions
Python expressions allow us to put Python code directly in the template. This is not meant for long, complex expressions.

```
<tal:block define="my_var python:foo and
                   bar or 'baz'" />
```

String Expressions
With string expressions, we can add static text to dynamic content or combine multiple variables

```
<div tal:define="itemNum string:row-
                 ${repeat/item/number};"
     tal:attributes="class string:itemRow
                     ${itemNum};">
```

TAL's attributes, in order of execution:

define
Defines variables to be used within the tag where defined.

condition
If `False`, the tag and everything in it will not render.

repeat
Repeat over a set of items. This has several properties available: `index`, `number`, `even`, `odd`, `start`, `end`, `length`. For example, `repeat/item/odd` will return `True` or `False`, depending on where you are in the repeat.

content
Content to display inside the tag.

replace
Content to display in place of the tag.

attributes
Allows you to put dynamic content inside of normal attributes like `href` and `src`.

omit-tag
Takes a condition. If it equates to `True`, the tag will not render, but everything inside it will.

For rich text, place in the template using structure: `<div tal:replace="structure context/getText">`. This will properly display the HTML instead of rendering it as text.

Plone's breadcrumb template has a good set of TAL statement examples. Here is a simplified version:

```
<span tal:repeat="crumb breadcrumbs"
      tal:attributes="id string:breadcrumbs-
                      ${repeat/crumb/number}">
  <tal:item tal:define="url crumb/absolute_url;
                        title crumb/Title">
    <a tal:omit-tag="not: url"
       tal:content="title"
       tal:attributes="href url">
          crumb
    </a>
    <span class="breadcrumbSeparator"
          tal:condition="not: repeat/crumb/end">
           &gt;</span>
  </tal:item>
</span>
```