World Food Programme

Drupal Standards & Best Practice

A guide to developing Drupal modules and themes for WFP

Andrew Holgate
Tuesday, February 28, 2012
Version 1.0

Table of Contents

Development Environment 3

- Development configuration and logging 3
- Recommended Development Tools 3

Drupal Theme Development 4

- Anatomy of a .tpl.php file 4
  - Example of node.tpl.php from Drupal core 4
- Theme preprocessor functions 5
- Overridding templates 5

Drupal Module Development 6

- Template files 6
- CHANGELOG 6
- When not to use PHP 6
- Module Dependencies (`.info`) 6
- Theme hook functions 6
- Translatable text 6
- Comments and formatting 7

Files directory 7

Coding Standards and Validation 7

- Drupal Standards 7
- Naming Conventions 7

CSS and JavaScript 8

Optimisation 8

- Caching 8
- Views 8
- Session data 8

Contributed Modules (drupal.org) 8

Quality Assurance and Testing 9

Deployment 9

- Deployment Dry-run 9

Training 9

- Technical installation and setup 9
- Backend training 9

Development Environment

The development environment must be identical to the WFP production system used for the platform, including:

- PHP version, php.ini configuration, additional modules installed and their versions
- MySQL version + configuration
- Apache version and additional modules
- Operating system and version

Development configuration and logging

The configuration of PHP must be identical to the production system with the exception of the following:

```
error_reporting = E_ALL | E_STRICT
display_errors = On
short_open_tag = Off
register_globals = Off
magic_quotes_gpc = Off
allow_call_time_pass_reference = Off
```

Regarding database configuration, slow query logging must be enabled with > 1 second being considered. Any new development must not add any additional slow queries to the log file but in the case that they do, a list of all database slow queries must be provided in a document which outlines steps taken to try to avoid the long queries.

Configuration examples:

```
slow_query_log          = 1
slow_query_log_file     = /var/log/mysql/mysql-slow.log
long_query_time         = 1
```

If you have not received the necessary configuration files and information, please contact WFP before the development cycle begins.

Recommended Development Tools

The following Drupal-specific development modules are highly recommended to be used during development:

- Coder[1] — Essential for WFP code review acceptance.
- Devel[2] — Highly recommended for all development
- Theme Developer[3] — Highly recommended for theme development
- Performance Logging and Monitoring[4] — Highly recommended for performance testing
- Schema[5] — Recommended for database verification
- Sitedoc[6] — Site documentation and unused themes, blocks ,etc

[1] http://drupal.org/project/coder
[2] http://drupal.org/project/devel
[3] http://drupal.org/project/devel_themer
[4] http://drupal.org/project/performance
[5] http://drupal.org/project/schema
[6] http://drupal.org/project/sitedoc

An excellent example of good theming practice can be found in this article: Theming Best Practices, Garland get a clean-up[7] - in the case of WFP development, the *template.php* should be used as little as possible and instead a custom theme support module should be used for logic.

Anatomy of a .tpl.php file

For an example of best templating practice, see the World Food Programme document entitled **Drupal Template Case Study**. If you were not supplied this document, please request it.

***.tpl.php must be used just for what templates are made for: markup**

Template files should be small, clean and only contain the following:

- Verbose comments
- XHTML
- Short, basic PHP functions, such as:
  - print
  - foreach
  - if-else

Avoid function calls directly in a *.tpl.php file. The *.tpl.php files should be used explicitly for markup, and just printing out variables when necessary. Function calls must be made in modules and the results passed through to the template.

Logic should never appear in a .tpl.php, it should be in a custom theme support module instead. In other words, creating arrays, imploding and function calls should never appear in a .tpl.php. Unlike the MVC (Model-View-Controller) architectural pattern which does not forbid direct access to the Model from the Views, the PAC (Presentation-Abstraction-Control) architectural pattern (which Drupal is based on) explicitly forbids doing this: the Control layer (i.e. controller) must guarantee that all the necessary variables which generation require communication with the Abstraction layer (i.e. model) will be prepared and passed to the Presentation (i.e. view) layer.

Verbose comments should be added to the beginning of each template file, such as is found in all of the Drupal code template files. In fact, the Drupal core template file comments can be used as a starting point for customized template files.

Example of node.tpl.php from Drupal core

```
<?php

/**
 * @file node.tpl.php
 *
 * Theme implementation to display a node.
 *
 * Available variables:
 * - $title: the (sanitized) title of the node.
 * - $content: Node body or teaser depending on $teaser flag.
 * - $picture: The authors picture of the node output from
 *   theme_user_picture().
 * - $date: Formatted creation date (use $created to reformat with
 *   format_date()).
 * - $links: Themed links like "Read more", "Add new comment", etc. output
 *   from theme_links().
 * - $name: Themed username of node author output from theme_username().
 * - $node_url: Direct url of the current node.
 * - $terms: the themed list of taxonomy term links output from theme_links().
 * - $submitted: themed submission information output from
```

[7] http://www.lullabot.com/articles/theming-best-practices-garland-gets-a-cleanup

```
 *   theme_node_submitted().
 *
 * Other variables:
 * - $node: Full node object. Contains data that may not be safe.
 * - $type: Node type, i.e. story, page, blog, etc.
 * - $comment_count: Number of comments attached to the node.
 * - $uid: User ID of the node author.
 * - $created: Time the node was published formatted in Unix timestamp.
 * - $zebra: Outputs either "even" or "odd". Useful for zebra striping in
 *   teaser listings.
 * - $id: Position of the node. Increments each time it's output.
 *
 * Node status variables:
 * - $teaser: Flag for the teaser state.
 * - $page: Flag for the full page state.
 * - $promote: Flag for front page promotion state.
 * - $sticky: Flags for sticky post setting.
 * - $status: Flag for published status.
 * - $comment: State of comment settings for the node.
 * - $readmore: Flags true if the teaser content of the node cannot hold the
 *   main body content.
 * - $is_front: Flags true when presented in the front page.
 * - $logged_in: Flags true when the current user is a logged-in member.
 * - $is_admin: Flags true when the current user is an administrator.
 *
 * @see template_preprocess()
 * @see template_preprocess_node()
 */
?>
<div id="node-<?php print $node->nid; ?>" class="node<?php if ($sticky) { print ' sticky'; } ?><?php if (!$status) { print ' node-unpublished'; } ?> clear-block">

<?php print $picture ?>

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

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

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

  <div class="content">
    <?php print $content ?>
  </div>

  <?php print $links; ?>
</div>
```

Notice the **verbose comments** and only **print** and the use of **simple code logic**.

Theme preprocessor functions

Preprocessor functions should be used extensively when variables need to be passed to .tpl.php files. This way, business logic can be performed inside of modules, with only the resulting variable(s) being passed through to the .tpl.php files, avoiding logic in the template files.

Overridding templates

When template files are being overridden in the theme, they should be put into a directory called “templates”, at the root of the custom theme directory.

The directory should be structured as follows:

```
/themes/wfp/templates/
                      /block/
                      /node/
                      /page/
                      /views/
                      /other/
```

Drupal Module Development

Template files

Template files (`.tpl.php`) related to the custom module should be in the root directory of the module. They can then be added to the Drupal templating system using `hook_theme()`[8] and also be passed custom theme variables.

CHANGELOG

Each custom module must contain the file `CHANGELOG.txt` which contains a list of all the changes between module release versions. Each modification to any file inside of the module should have a summary of the modification added to the `CHANGELOG.txt` file.

When not to use PHP

PHP code should only ever appear in files and **never in the database**.

So, PHP code should never be used in textfields inside Views, Panels, Webform, nodes, blocks etc. even if the PHPFilter module is enabled.

If beginning a new Drupal project, **never** activate the core *PHP Filter* module or any other module that allows PHP to be evaluated from textfields.

Here is a list of strong reasons not to use PHP in text fields: Downside of using PHP code in textfields[9]

Module Dependencies (`.info`)

The `module.info` file should be updated to reflect the version changes (eg. 6.x-1.1) and dependencies as changes to the module are made.

Any calls to modules functions require that the module be added as a dependency to the .info file. It should not be assumed that optional core modules are installed (e.g. taxonomy, path, etc.)

The Features module, by default, suggests the dependent modules, each of these suggestions should be verified if it is an actual dependency or not.

Theme hook functions

Already existing theme hook functions should be used whenever possible, rather than creating HTML elements inside modules. For example, use:

For image elements (`<img>`), use: (always include the alt and title parameters)

```
theme('image', 'files/image.png', 'An Image', 'Description of an image');
```

For anchor elements (`<a>`), use: (always include the title option, and an appropriately named class and id when necessary)

```
l(t('Label'), 'node/123', array('attributes' => array('class' => 'link', 'id' => 'label', 'title' => 'Title')));
```

Translatable text

Human-readable text that will be displayed somewhere within a page must be run through the t()[10] function.

The developers must use the placeholder parameters[11] of the t() function whenever using a dynamic string.

[8] http://api.drupal.org/api/function/hook_theme
[9] http://drupal.stackexchange.com/questions/2509/what-are-the-downsides-of-using-custom-php-code-in-blocks-nodes-views-args#answers-header
[10] http://api.drupal.org/api/drupal/includes%21common.inc/function/t/6

Comments and formatting

The developers should be well aware of and develop inline with the Drupal best practices[12] guideline and importantly, the Drupal coding standards[13]. These guidelines should be followed strictly.

Files directory

The files directory should contain sub-directories which allow for files to be logically divided.

For example:

```
/files/images/
/files/pdf/
/files/fr/images/       (for i18n installations)
```

On sites where users will be allowed to upload files and it is foreseen that the number of files on the server will increase over time, the module FileField Paths[14] should be used and an appropriately named directory in the *files* directory should be used.

Coding Standards and Validation

Drupal Standards

The Coder module[15] should be used as a guideline to gauge if the custom code has been developed using the Drupal coding standards.

When submitting development for approval, a document should be provided with the results of running a code review using the following parameters (only applied to the modules / themes that were worked on during the development):

- Drupal Coding Standards
- Drupal Commenting Standards
- Drupal SQL Standards
- Drupal Security Checks
- Internationalization
- Include files (inc | php | install | test)
- "Normal" severity level.

A WFP technical staff member will also perform a code review to verify if the Best Practices outlined in this document have been followed.

Naming Conventions

PHP variable names should be descriptive and easily readable to determine what kind of data it contains.

CSS selector names (both ID's and classes) should use semantic names to identify them. The supplier should contact WFP regarding the specific naming convention.

[11] http://drupal.org/node/322732
[12] http://drupal.org/best-practices
[13] http://drupal.org/coding-standards
[14] http://drupal.org/project/filefield_paths
[15] http://drupal.org/project/coder

CSS and JavaScript

CSS and JS files should always be added using the Drupal functions `drupal_add_js()` and `drupal_add_css()` respectively which allows them to be aggregated and optimized. They should never be added directly inside template files.

Inline CSS and JS should never be used when it can be put into an external file.

JavaScript and CSS should also conform to standard practices regarding formatting, layout and logic.

As we aggregate both JavaScript and CSS, logical separation can be made by creating separate files when necessary. Adding **all** CSS to *`style.css`* in the theme directory eventually becomes unmanageable so logical separation is encouraged.

CSS selectors (id's and classes) should be used frequently and appropriately.

Optimisation

Optimisation must be considered even from the design stages of development and not seen as an afterthought, once development has been completed. Some grave examples have been discovered which would mean having to completely redesign whole sections of the platform in order to get it to work with caching.

Caching

All custom code must function when all forms of caching is active, including: Page caching (including Pressflow's cache lifetime options), Block caching, URL Path Caching (Pressflow), Views caching, Panels caching, etc. All code must also work with MemCache as well as with Varnish and other reverse proxies.

All code must also work with JS and CSS optimization activated.

Views

When necessary, a View should be created and organized to contain appropriate, logical groups of displays (for example, by section or by functionality).

Session data

Anonymous PHP sessions must not be used (ie. using the *`$_SESSION[]`* associative array global variable), unless approved by WFP.

Contributed Modules (drupal.org)

The golden rule to using contributed modules is to keep it simple and *only install the essential modules required to achieve the functionality requested*.

Contributed modules should be analysed thoroughly to assess if they are essential (e.g. a module where only 10% functionality is used, is not essential. Instead, write a new module to address the specific needs, taking into consideration performance and scalability issues. Contributed modules which leverage node functionality potentially can create an unacceptable database overhead when a system needs to handle a lot of users.) Developers must be highly aware of the performance / security / migration pitfalls of using too many contributed modules (see: The Drupal contributed modules "open buffet binge" syndrome[16].)

[16] http://2bits.com/articles/server-indigestion-the-drupal-contributed-modules-open-buffet-binge-syndrome.html

Additional contributed modules from drupal.org can only be installed once approved by WFP. The vendor must be able to justify the installation of the additional module. Approval is also required for contributed modules which are already available on the repository but are not active.

Release Candidate and stable versions of modules are permitted to be installed, unless there is a known vulnerability or major issue with the module. Development, alpha and beta versions of modules are not permitted to be installed.

Quality Assurance and Testing

The vendor has to carry out full Quality Assurance (QA) rounds to ensure the deliverables are as requested and that they are completely bug and error free.

WFP will carry out Acceptance Testing (AT) of the deliverables to ensure that they meet the functionality requirements. Invoices will not be paid until all WFP Acceptance Tests have been passed.

The vendor must provide a 12-month bug warranty, fixing identified bugs within a reasonable time period, free of charge.

Deployment

The supplier must package the changes into an existing Feature when possible and if not, into a new Feature[17].

A feature is a collection of Drupal entities which taken together satisfy a certain use-case. Importantly, the supplier should add custom code (e.g. custom hook implementations, other functionality, etc.) to your feature in *myfeature.module* as you would with any other module. All Features must contain the correct dependencies (in the .info file) and be 100% self-contained.

All deployment packages (whether Features or not) must have an *INSTALL.txt* document which clearly outlining the step-by-step installation procedure for the new functionality.

Deployment Dry-run

The supplier must perform a dry-run of the integration of the new code / Feature using an exact copy of the production site. Once the supplier is completely satisfied with the deployment and can verify that the supplied *INSTALL.txt* document is correct, the package should only then be sent to WFP for deployment.

Training

Technical installation and setup

Sufficient time must be made available to work alongside WFP to successfully complete the deployment on a staging server as well as on the production server.

Backend training

If the development modifies or adds functionality to the backend interface (additional features, menu options, etc.) then a clear, well written user manual must be provided or the current user manual updated to represent the changes.

[17] http://drupal.org/project/features