


# Drupal Module Development

Or: How I Learned to Stop Worrying and Love the Module

Alastair Moore & Paul Flewelling

boost new media 

# What does a module do?

- ▶ Core modules provides functionality
- ▶ Contributed modules extends functionality

# Examples of modules

- ▶ Core required modules:
  - ▶ System, Filter, Node, Block, User
- ▶ Core optional modules:
  - ▶ Taxonomy, Menu, Comment, Search
- ▶ 3rd party modules:
  - ▶ Views, Panels, CCK, Devel

# Hooks

- ▶ Can be thought of as internal Drupal events
- ▶ Allow modules to interact with the Drupal core
- ▶ Events include:
  - ▶ Creation, deletion of nodes and users
  - ▶ Logging in, logging out
- ▶ Hooks can also create and alter menus and forms

# Hook examples

- ▶ hook\_user
  - ▶ login, logout, insert, view, update
- ▶ hook\_block
  - ▶ view, save, configure, list
- ▶ hook\_form\_alter
- ▶ hook\_insert
- ▶ hook\_comment
  - ▶ insert, update, view, publish, unpublish

hook\_user – allows the module to react when operations are performed  
hook\_block – declares a block or set of blocks  
hook\_form\_alter – performs alterations to a form or forms before it's rendered

# Useful modules

- ▶ Devel ([drupal.org/project/devel](http://drupal.org/project/devel))
- ▶ Admin Menu ([drupal.org/project/admin\\_menu](http://drupal.org/project/admin_menu))
- ▶ Drush ([drupal.org/project/drush](http://drupal.org/project/drush))
- ▶ Any others?

Module builder – still in beta. For the lazy.

Drush – command line utility. Drupal Shell. Installing modules, clearing cache, run SQL commands on the database

# Our first module

- ▶ 'Hello World' block module
- ▶ Two files required for a module:
- ▶ helloworld.info
- ▶ helloworld.module
- ▶ (Optionally) helloworld.install

# Our first module

Where should I put my module?

- ▶ Everyone else's:
  - ▶ `/sites/drupal-south.local/modules/contrib`
- ▶ Yours (in development):
  - ▶ `/sites/drupal-south.local/modules/dev`
- ▶ Yours (after development):
  - ▶ `/sites/drupal-south.local/modules/boost`

# helloworld.info

```
; $Id$  
name = Hello World  
description = Creates a Hello World block  
package = Dev  
core = 6.x
```

# helloworld.module

```
<?php
function helloworld_block($op = 'list', $delta = 0, $edit = array())
{
    switch($op)
    {
        case 'list':
            $block[0]['info'] = t('Hello World');
            return $block;
            break;
    }
}
```

Only first <?php declaration needed. Removes possibility of errors occurring due to unwanted whitespace.  
Edit contains submitted form data from block config form.

# helloworld.module

```
case 'view':  
    $block['subject'] = t('Hello World!');  
    $content = 'Hello world!';  
    $block['content'] = $content;  
    return $block;  
    break;
```

# helloworld.module

```
case 'configure':  
    $form['helloworld_count'] = array(  
        '#type' => 'textfield',  
        '#title' => t('Number of Hello Worlds to display'),  
        '#default_value' => variable_get('helloworld_count', 1)  
    );  
    return $form;  
    break;
```

# helloworld.module

```
case 'save':  
    variable_set('helloworld_count', $edit['helloworld_count']);  
    break;
```

# helloworld.module

```
case 'view':
```

```
    $block['subject'] = t('Hello World!');
```

```
    $count = variable_get('helloworld_count', 1);
```

```
    for ($i = 0; $i < $count; $i++)
```

```
    {
```

```
        $content .= '<li>Hello world!</li>';
```

```
    }
```

```
    $block['content'] = $content;
```

```
    return $block;
```

```
    break;
```

# So far, so good?

## Any questions?

# Theming a module

- ▶ Register a theme
- ▶ Implement a theme function
- ▶ Applying the theme
- ▶ Utilise external theme templates

# Registering a theme

```
function helloworld_theme()  
{  
    return array(  
        'helloworld_show' => array(  
            'arguments' => array('content' => NULL),  
        ),  
    );  
}
```

# Implementing theme

```
function theme_helloworld_show($content)
{
  $output = '<ul>$content</ul>';
  return $output;
}
```

# Applying the theme

```
case 'view':  
    $block['subject'] = t('Hello World!');  
    $count = variable_get('helloworld_count', 1);  
    for ($i = 0; $i < $count; $i++)  
    {  
        $text .= '<li>Hello world!</li>';  
    }  
    $content = theme('helloworld_show', $text);  
    $block['content'] = $content;  
    return $block;  
    break;
```

# External theme file

```
function helloworld_theme()  
{  
    return array(  
        'helloworld_show' => array(  
            'arguments' => array('content' => NULL),  
            'template' => 'helloworld_show'  
        ),  
    );  
}
```

# helloworld\_show.tpl.php

```
<p>
```

```
    This is being called from the external file.
```

```
</p>
```

```
<ul>
```

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

```
</ul>
```

# Using deltas

```
function helloworld_block($op = 'list', $delta = 0, $edit = array())
{
  switch($op)
  {
    case 'list':
      $blocks[0]['info'] = t('Hello World');
      $blocks[1]['info'] = t('Goodbye Cruel World');
      return $blocks;
      break;
  }
}
```

# Using deltas

```
case 'view':  
  switch ($delta) {  
    case 0:  
      $block['subject'] = t('Hello World!');  
      $content = 'Hi! :-)';  
      break;  
    case 1:  
      $block['subject'] = t('Goodbye Cruel World!');  
      $content = 'Bye... :-(';  
      break;  
  }  
  $block['content'] = $content;
```

# Persisting Our Message

## Databases and Admin Forms

- ▶ 1. Create a table in the database that will hold our message
- ▶ 2. Create an administration form so we change the message

We decide that we want to store the hello world message in the database. This would usually be used for a more sophisticated problem, but I'd like to demonstrate the use of the database and the Form API to create an administration form.

So, we need to 1. Create a table in the database to hold our hello world message and 2. Create an administration form so we can change the message

# Modifying the database

- ▶ Create a new file called helloworld.install
- ▶ Define the schema using the hook hook\_schema
- ▶ Install the schema using hook\_install

To modify the database we create a new file called helloworld.install

We then define the schema of our database changes using the hook hook\_schema

We then install the schema using the hook hook\_install

# Define the Schema

## hook\_schema

```
function helloworld_schema() {  
  $schema['helloworld'] = array(  
    'description' => t('Helloworld Text'),  
    'fields' => array(  
      'helloworld_text' => array(  
        'description' => t('Text to say to the World'),  
        'type' => 'varchar',  
        'length' => 128,  
        'not null' => TRUE,  
      ),  
    );  
  return $schema;  
}
```

<-- DEFINES A TABLE

<-- DEFINES A FIELD

The hook\_schema looks something like this.

Here we are describing a new table called helloworld and a single field to store the message.

In fact our schema is slightly more complicated than it is shown here. We will also be defining a display\_count and unique identifier.

# hook\_install

## Install the schema

```
function helloworld_install() {  
  drupal_install_schema('helloworld');  
}
```

# hook\_uninstall

## Uninstall the schema

```
function helloworld_uninstall() {  
  drupal_install_schema('helloworld');  
}
```

And for good measure, the `hook_uninstall`. This removes the table (and related data) from the database. This is run when using “Uninstall” in the modules area of the admin console.

So we end up with a file that looks like this **SHOW ACTUAL SCHEMA CODE**

# Creating the Admin Form

- ▶ New file called helloworld.admin.inc
- ▶ Create the form using the Form API

So, we've created our database, now it's time to create the administration form.

# Creating the Form - 1

```
function helloworld_admin_edit() {  
  $form = array();  
  $form['helloworld'] = array(  
    '#type' => 'fieldset',  
    '#title' => t('Helloworld'),  
    '#description' => t("The message to send to the World")  
  );  
  $form['helloworld']['message'] = array(  
    '#title' => t('Helloworld Message Text'),  
    '#type' => 'textfield',  
    '#description' => t('The Helloworld Message Text'),  
    '#size' => 30,  
    '#maxlength' => 30,  
  );  
}
```

<-- DEFINES THE FORM

<-- DEFINES 1ST FIELD

Here we're building up the form using the Form API to define the form and its fields.

For example the first chunk of code here defines the form, its name and its purpose.

The second chunk represents the field the message will be typed into, currently limited 30 characters.

# Creating the Form - 2

(continued)

```
$form['helloworld']['display_count'] = array(
  '#title' => t('Display Count'),
  '#type' => 'textfield',
  '#description' => t('The number of times to display'),
  '#size' => 2,
  '#maxlength' => 2,
);
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Update'),
);
return $form;
}
```

<-- DEFINES THE  
DISPLAY COUNT FIELD

<-- SUBMIT BUTTON

Another field defined here allows us to record the `display_count`, i.e. how many times the message will be repeated in the block.

And finally we define the form's submit button

# Form Validation

## Check the data is legit

```
function helloworld_admin_edit_validate($element, $form_state) {  
  $message = $form_state['values']['message'];  
  if (empty($message)) {  
    form_set_error('message', t('Please enter a message for the World'));  
  }  
  $display_count = $form_state['values']['display_count'];  
  if (!is_numeric($display_count)) {  
    form_set_error('display_count', t('Please enter a display count'));  
  }  
}
```

Note the name of the method, it specifically validates module Helloworld's admin\_edit form

1. Validates that the message isn't empty
2. Validates the display\_count is a number

# Submit

## Update the database

```
function helloworld_admin_edit_submit($form, &$form_state) {  
  $result = db_query("UPDATE {helloworld}  
    SET message = '%s', display_count = '%d'  
    WHERE {helloworld}.hwid = 1",  
    $form_state['values']['message'],  
    $form_state['values']['display_count']);  
  if ($result) drupal_set_message(t('The Helloworld text has been updated.));  
}
```

# Permissions

## hook\_perm

```
function helloworld_perm() {  
    return array('change message', 'view message');  
}
```

This gives us to options under `/users/permissions`, allowing us to determine who can change the message and who can view the message.

**\*SHOW USER / PERMISSION ADMIN PAGE\***

# Menus

## hook\_menu

```
function helloworld_menu() {  
  $items['admin/settings/helloworld/settings'] = array(  
    'title' => 'Hello World settings',  
    'page callback' => 'drupal_get_form',  
    'page arguments' => array('helloworld_admin_edit'),  
    'access arguments' => array('change message'),  
    'file' => 'helloworld.admin.inc',  
    'type' => MENU_NORMAL_ITEM,  
  );  
  return $items;  
}
```

<-- FORM API

<-- PERMISSION

hook\_menu is placed in the module file, it defines an entry in Drupals menu hierarchy so that we can access the administration form.

**\*SHOW MENU\***

# Updating the *Module*

- ▶ Now need to retrieve data from the database
- ▶ Modify the block code to display the data

We've completed the major steps, now all we need to do is

1. Retrieve the data from the database  
and
2. Modify the block code display the data

# Retrieve Message

Query database and return data objects

```
function _helloworld_get_message() {  
    $result = db_query('SELECT {helloworld}.message,  
{helloworld}.display_count FROM {helloworld} WHERE {helloworld}.hwid = 1');  
    $rows = array();  
    while($row = db_fetch_object($result)) {  
        $rows[] = $row;  
    }  
    return $rows;  
}
```

This code also goes in the module file.

It executes a query to retrieve the data from the helloworld table.

It returns a database object

# Modify Block

## Update hook\_block view to use data objects

```
case 'view':  
    $block['subject'] = t('Hello World!');  
    $hw_vars = _helloworld_get_message();  
    for($i = 0; $i < $hw_vars[0]->display_count; $i++)  
    {  
        $content .= "<p>".$hw_vars[0]->message."</p>";  
    }  
    $block['content'] = $content;  
    return $block;  
    break;
```

Here we see the ‘view’ case of the hook\_block method we defined earlier.

It has been modified to call call the helloworld\_get\_message method that queries the database.

It then formats the data and places it in the \$content variable ready for display by the block.

**\*SHOW THE FINISHED PRODUCT FROM END TO END\***

# Further resources

- ▶ [api.drupal.org](http://api.drupal.org)
- ▶ Pro Drupal Development (2nd Edition)
- ▶ Learning Drupal 6 Module Development
- ▶ [www.lullabot.com](http://www.lullabot.com)