api.backdropcms.org

Converting modules to Backdrop from Drupal 7

Backdrop's codebase is similar to Drupal 7 and generally needs very few changes to convert. To make things even easier, Backdrop includes a Drupal compatibility layer which should enable Drupal functions (even those with the format '`drupal_XXX`', such as `drupal_set_message`), to just work, for now. However, Backdrop still has significant advantages over Drupal 7, and it would make sense to fully port your module rather than depend on the compatibility layer.

This guide suggests a process for converting Drupal 7 modules to Backdrop, and the common issues which might arise during the conversion. If your module is another Drupal version, this would be mostly out of the scope of this guide; we would recommend you follow Druapl's module conversion guidelines to upgrade (or downgrade) to Drupal 7 before continuing.

A useful video by @Quiksketch demonstrates most of the main issues in this document

Let's dive in - modifying your module's .info file

The developer documentation for Backdrop modules is here. (https://api.backdropcms.org/modules). For comparison, here is a standard Drupal module info file:

```
name = Book
description = Allows users to create and organize related content in an outline.
package = Core
version = VERSION
core = 7.x
files[] = book.test
configure = admin/content/book/settings
stylesheets[all][] = book.css

; Information added by Drupal.org packaging script on 2014-11-19
version = "7.34"
project = "drupal"
datestamp = "1416429488"
```

and here is Backdrop:

```
name = Book
description = Allows users to create and organize related content in an outline.
package = Core
version = BACKDROP_VERSION
backdrop = 1.x
dependencies[] = node
configure = admin/content/book/settings
```

The first and main change is that the version key:value pair is now '`backdrop: 1.x`'. The accepted .info file contents are essentially unchanged apart from this.

Organization of module files

If your module contains tests, information about tests now in `module.tests.info`. Otherwise your modules file organization can remain unchanged.

However there are a few recommended standards for locations of module files. As stated in the API documentation, `.theme.inc`, `.admin.inc`, and `.pages.inc` files are recommended to be kept in the root of the module folder to allow ease of recognition of the way the module likely works. This makes it more obvious that the module provides theme functions, has an admin page, and probably provides display pages in some way. If there are multiple of these files however, it is reasonable to move these files to an includes folder in the module root.

Javascript files are always kept in a `/js` folder, templates in a `/templates` folder, and tests in a `/tests` folder.

Otherwise, thats about it. If the module is now viewed on the modules page of a Backdrop it could now be enabled, and possibly might even work normally in simple cases. But there are a few API changes in Backdrop which may need to be dealt with.

Working through common API changes.

Configuration management (CMI)

Drupal 7 stores module configuration in variables, which are manipulated by the function trio of `variable_get()`, `variable_set()`, and `variable_del()`. These values are stored in the database in the '`variables`' table.

In Backdrop, configuration has moved to being stored in text files, called configuration (or config) files, with a .json extension. The Config module provides the corresponding `config_get()` and `config_set()` manages the retrieval and writing to disk of config. Config files are saved in `BACKDROP_ROOT/files/config_xxxx`, where xxxx is a long alphanumeric string.

For the immediate future, the '`variable_`' functions still exist and work in Drupal, but conversion to '`config_`' is recommended.

Deciding on the config storage file(s)
To convert variables to config, first identify all the module's '`variable_`' functions; use your editor search or grep - the aim initially is to get an idea of the number and type of configuration your module requires. To store config, the module will need its own config file, commonly designated as '`my_module.settings.json`'; this should be sufficient for the majority of modules. If on searching, it seems more reasonable to group settings into more than one single category however, more than one config file is appropriate. For example, Node module settings are saved by node type, for example `node.type.article.json`, and `node.type.page.json`. So for a new Drupal to Backdrop port for a module with complex configuration, `my_module.first_set_of.settings.json`, and `my_module.second_set_of.settings.json` may also be totally appropriate.

Changing the code
Once this has been decided, every instance of a '`variable_`' function will now need to be replaced by config. The standard method is as follows:

Instead of:

```
// Get a variable
$variable = variable_get('foo', 'bar');
```

```
// Set a variable
variable_set('foo', $variable);
```

Use CMI:

```
// Get a config setting
$variable = config_get('my_module.settings', 'foo');

// Set a config setting
config_set('my_module.settings', 'foo', $variable);
```

The above will read and write to a JSON file called `my_module.settings.json` in the active config directory.

An alternative method is to create a config object and use the object's get and set methods.

```
// Get the full config object
$config = config('my_module.settings');
// Get individual settings
$value_foo = $config->get('foo');
$value_next = $config->get('next');
```

This is useful for example in form arrays, where '`#default_value`' is being retrieved for many form elements. If every element used `config_get`, this requires a separate read of the config file for each instance of `config_get`. The object/method approach limits this to one read.

Similarly, in a form submit function, where config is being saved, use:

```
<?php
// Get the full config object
$config = config('my_module.settings');

// Set individual config settings
$config->set('setting_one', $form_state['values']['value_one']);
$config->set('setting_two', $form_state['values']['value_two']);

// Save
$config->save();
?>
```

Let Backdrop know about your module's config
Modules also need to declare config files in `hook_config_info()`.

```
<?php
/**
* Implements hook_config_info().
*/
function contact_config_info() {
  $prefixes['my_module.first_set_of.settings'] = array(
    'label' => t('First set of settings'),
    'group' => t('My Module'),
  );
  $prefixes['my_module.second_set_of.settings'] = array(
    'label' => t('Second set of settings'),
    'group' => t('My Module'),
```

```
  );
  return $prefixes;
}
?>
```

Create your config defaults
The function variable_get() has the useful capacity to declare default values; writing `$setting = variable_get('my_setting', 'red')` allows the developer to initialize the value 'red' if 'my_setting' variable was not previously saved. This is not available to CMI, and therefore config defaults need to be explicitly set at module enabling. This can be done in hook_install(), in an `.install` file.

The simplest method is in four steps:

First step: set ALL your modules config settings manually in hook_install.

```
// Save all settings to a file called 'my_module.settings'
$config = config('my_module.settings');
$config->set('setting_one', 'value_one');
$config->set('setting_two', 'value_two');
... (all other config)

$config->save();
```

Step two: Enable the module
Enabling the module causes a file `my_module.settings.json` to be saved in the default config folder in `files`.

Step three: now copy this my_module.settings.json file and save it in a `config` folder in the root of your module folder.

```
BACKDROP_ROOT/modules
    my_module
        my_module.info
        my_module.module
        config
            my_module.settings.json
```

Step four: delete the config functions from hook_install() which you added in step one. They are no longer needed. On future module installs, Backdrop will automatically recognize the `config` folder in your module root and will automatically copy the `my_module.settings.json` file from your module's config folder to the default config folder!

At this stage the module should be nearly completely converted to CMI; new installs will use CMI exclusively for storing config. However, existing users with this module installed on a Drupal site and who wish to convert to Backdrop will still need a process to convert exisiting settings stroed as variables to the config system.

Create an upgrade path
Backdrop, like Drupal uses the hook_update_N() system for upgrades and updates. For initial porting to Backdrop 1.x, hook_update_10NN could be used, where NN is any two digit number, such as `hook_update_1000`.

In this update hook, retrieve the values of existing variables, and store them as config, then delete the variables. For example:

```
<?php
```

```
/**
 * Move book settings from variables to config.
 */
function book_update_1000() {
  // Migrate variables to config.
  $config = config('book.settings');
  $config->set('book_allowed_types', update_variable_get('book_allowed_types',
array('book')));
  $config->set('book_child_type', update_variable_get('book_child_type', 'book'));
  $config->set('book_block_mode', update_variable_get('book_block_mode', 'all
pages'));
  $config->save();

  // Delete variables.
  update_variable_del('book_allowed_types');
  update_variable_del('book_child_type');
  update_variable_del('book_block_mode');
}
?>
```

Replacing system_settings_form()

A commonly related conversion is the use of `system_settings_form()`.

> "This function adds a submit handler and a submit button to a form array. The submit function saves all the data in the form, using variable_set(), to variables named the same as the keys in the form array."

Since variables are no longer used, `system_settings_form()` is deprecated. Instead,

Old:

```
<?php
function my_settings_form($form, $form_state) {
  $form['my_setting'] = array(
    '#type' => 'textfield',
    '#title' => t('My setting'),
    '#default_value' => variable_get('my_setting', ''),
  );
  return system_settings_form($form);
}
?>
```

New:

```
<?php
function my_settings_form($form, $form_state) {
  $config = config('my_module.settings');

  $form['my_setting'] = array(
    '#type' => 'textfield',
    '#title' => t('My setting'),
    '#default_value' => $config->get('my_setting'),
  );

  // Add a submit button
```

```
  $form['actions']['#type'] = 'actions';
  $form['actions']['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Save configuration'),
  );

  // Return the form!
  return $form;
}

// Add a submit function to process config
function my_settings_form_submit($form, &$form_state) {
  $config = config('my_module.settings');
  $config->set('my_setting', $form_state['values']['my_setting']);
  $config->save();
  backdrop_set_message(t('The configuration options have been saved.'));
}
?>
```

Classed entities

Several entities in Backdrop are now classed objects which extend the Entity Class. **Users**, **nodes**, **comments**, **files**, **taxonomy terms** and **vocabularies** have been converted to extend the new base class. Therefore programmatic creation of these objects has changed in Backdrop. For example:

Old

```
$node = new stdClass();
$node->type = 'blog';
$node->title = 'New Title';
...
node_save($node);
```

New

```
$node = entity_create('node', array ('type' => 'blog'));
$node->uid = $user->uid;
$node->title = 'New Title';
$node->save();
```

Help module is gone

Finally an easy API change: help module is removed in favor of anticipating a more robust contrib help-related module (still to come). Most modules in Drupal 7 still have references to `hook_help()` however. These will not affect a module conversion in any way, since `hook_help()` will simply not be called. But these implementations can be removed.

Summary

These are a few of the common module changes which will be required for porting. However, other less common API changes exist. See the change records for a full list. If you find an unlisted API change, please report this on the Backdrop Issue Queue.

Happy converting!

‹ Creating modules up Creating layouts ›

Function, file, or topic *

Partial match search is supported

Search

Navigation

- API Reference

Resources

- Change Records
- Developer Documentation
  - Creating modules
    - Converting modules
  - Creating layouts
  - Creating themes
  - PHP Coding Standards
  - CSS Coding Standards
  - JS Coding Standards
  - Documentation Standards
- PHP Coding Standards
- CSS Coding Standards
- JavaScript Coding Standards
- Code Documentation Standards
- Submitting Pull Requests