The Drupal 8 API

The Drupal 8 API

Table of Contents

Chapter 1. Introduction

This book provides information for Drupal developers on the Drupal API, and is meant to act as a companion to the detailed class and function reference information that you can find on https://api.drupal.org (the official Drupal API reference site). The information here is of a tutorial and descriptive nature, and is built from AsciiDoc [http://asciidoc.org] source files that you can find in the "Core Docs" project on Drupal.org. (In contrast, the function/class reference information on https://api.drupal.org is built from specially-formatted comments embedded in the Drupal Core files themselves.)

Chapter 2. Configuration API

The configuration API provides a central place for modules to store configuration data. This can be simple configuration like your site name, or more complex information managed with configuration entities, such as views and content types.

Overview of Configuration (vs. other types of information)

In Drupal 8, there are several types of information:

Content	Information meant to be displayed on your site: article, basic page, images, files, etc.
Session	Information about individual users' interactions with the site, such as whether they are logged in. This is really "state" information, but it is not stored the same way so it's a separate type here.
State	Information of a temporary nature about the current state of your site. Examples: the time when Cron was last run, whether node access permissions need rebuilding, etc. See Chapter 3, *State API* for details of how to store this information.
Configuration	Information about your site that is not content and is meant to be more permanent, such as the name of your site, the content types and views you have defined, etc.

Deciding how to classify your information

It is not always clear how to decide whether a piece of information that your module will store should be classified as content, state, or configuration. Here are some guidelines:

Configuration vs. State	If your information would need to be deployed from your development server to your live server, it is probably configuration and not state information.
Configuration vs. Content	Think site builder vs. site editor. If a "site editor" role on the site would want to edit the information, it is probably content. If only a "site builder" role would want to have the power to edit the information, then it is probably configuration. But this is not an absolute rule.
Configuration vs. Content	Think about numbers. If you have a huge number of items, probably it is content. If you will only ever have a few, probably it is configuration.
Configuration vs. Content	Configuration tends to define "types of things", such as content types, taxonomy vocabularies, etc. Then each "thing" within the type is a piece of content: a content node, a taxonomy term, etc.

Simple Configuration vs. Configuration Entities

There are two overall types of configuration information. Simple configuration is used for single, global settings, such as the name of your site. Configuration entities are used for pieces of information that have multiple copies; for example, views, content types, etc.

Configuration Storage

Configuration information is stored in files and in the database.

Configuration File Format (YAML)

All configuration data is stored on-disk using YAML files.

Here is an example of a configuration file:

```
some_string: 'Woo kittens!'
some_int: 42
some_bool: true
```

Configuration can also be nested. Here is an example:

```
name: thumbnail
label: 'Thumbnail (100x100)'
effects:
  1cfec298-8620-4749-b100-ccb6c4500779:
    id: image_scale
    data:
      width: 100
      height: 100
      upscale: true
    weight: 0
    uuid: 1cfec298-8620-4749-b100-ccb6c4500779
```

See the section called “Configuration Schema and Meta-Data” for information on the schema for configuration files.

Configuration File Location for a Site

By default, when you install Drupal, the installer will create a randomly-named directory inside your public files directory for configuration. The name will start with *config_*, followed by a random hash string. Within this directory, the installer will create an *active* directory for your current live configuration, and a *staging* directory for configuration you are importing.

You can change the locations of your staging and active directories by editing your settings.php file.

Default Configuration for a Module

A module that provides default values for its configuration must put that configuration into YAML files in its *config* sub-directory.

If the module only needs basic Simple Configuration settings, all of the default configuration could go into one modulename.settings.yaml file. For more complex settings, you can separate your configuration into multiple files. Configuration Entities must each be put into their own YAML files, and they should be generated by having the module write out its configuration (don’t try writing them by hand).

Configuration Caching in the Database

The canonical storage for configuration for a site is the files in the *active* configuration directory defined in settings.php. However, by default, Drupal table. (Of course, you can override Drupal’s default cache system.)

Configuration Schema and Meta-Data

The configuration API includes support for a Kwalify Kwalify [http://www.kuwata-lab.com/kwalify/]-inspired schema/metadata language for configuration YAML files. Kwalify itself is written in Python and we needed slight adjustments in the format, so not all of the details of Kwalify are directly applicable, but it is pretty close.

@todo Put the rest of https://drupal.org/node/1905070 here.

Rest of Configuration API information

@todo Put the rest of the Config API section from https://drupal.org/node/1667894 here. and update the title and identifier for this section.

Chapter 3. State API

The State API provides a place for developers to store information about the system's state. A good example of state is the last time cron was run. This is specific to an environment and has no use in deployment. See the section called "Overview of Configuration (vs. other types of information)" for a more complete discussion of how state information differs from configuration information.

The State API is a simple system to store this information. Typical usage:

```
// Get a value:
$val = Drupal::state()->get('key');
// Get multiple key/value pairs:
$pairs = Drupal::state()->getMultiple($keys);
// Get all key/value pairs:
$collection = Drupal::state()->getAll();
// Set a value:
Drupal::state()->set('key','value');
// Set multiple values:
Drupal::state()->setMultiple($keyvalues);
// Set a value if not already set:
Drupal::state()->setIfNotExists('key','value');
// Delete a value:
Drupal::state()->delete('key');
```

See also:

- the section called "Updating Drupal 7 Variables to Drupal 8 State System"

Chapter 4. Updating to Drupal 8

This section contains topics about updating modules from Drupal 7 and other previous versions to Drupal 8.

Updating Drupal 7 Variables to Drupal 8 State System

Here is a guide for how to update Drupal 7 variables to the State system. Note that some variables should be updated to the configuration instead — see the section called “Overview of Configuration (vs. other types of information)” for more information.

Here are the steps to follow:

1. Within your conversion issue work, convert one variable at a time.
   - Determine the variable name to convert.
   - Grep the entire Drupal code base for the variable name and identify all instances that need to be updated.
2. Keep state identifiers short and concise. Generally these can probably stay the same as they were in Drupal 7, unless the old name was incorrect or confusing in some way.
3. The state system is initialized by calling the Drupal::state() function. Interact with the state system using the get(), set() and delete() functions. The Drupal::state() function returns a state object, so you can do:

```
$data = Drupal::state()->get('my_state_data');
```

4. When retrieving data, the state system does not provide a way to provide a default the way the old variable system did. However, you can provide a default when it returns FALSE, indicating that there is no data. A concise way to do this is:

```
$state = Drupal::state()->get('my_state') ?: 'Nothing there';
```

Note: If the boolean value of FALSE or the integer 0 are valid data for your state variable, then this will require special handling.

5. Here is a simple example of converting variables to state:

```
//  Drupal 7
variable_set('my_data', 'foo');
$data = variable_get('my_data', 'bar');
variable_del('my_data');

// Drupal 8
Drupal::state()->set('my_data', 'foo');
$data = Drupal::state()->get('my_data') ?: 'bar';
Drupal::state()->delete('my_data');
```

6. The variable name should be changed so that we can identify the module that creates it. The key should use the same namespace strategy as the configuration system. So for example:
   - cron_last becomes system.cron_last

- node_cron_last becomes node.cron_last
- menu_masks becomes menu.masks

7. The upgrade path needs to be determined. If the value needs to be maintained through the Drupal 7 to 8 upgrade it should be migrated. For example:

```
/**
 * Migrates install_task and install_time variables to State API.
 *
 * @ingroup state_upgrade
 */
function system_update_8022() {
  update_variables_to_state(array(
    'install_task' => 'system.install_task',
    'install_time' => 'system.install_time',
  ));
}
```

However if the value will be recreated through cache clears or naturally through the upgrade then the Drupal 7 variable should be deleted. For example:

```
/**
 * Delete drupal_js_cache_files variable.
 *
 * @ingroup state_upgrade
 */
function system_update_8023() {
  update_variable_del('drupal_js_cache_files');
}
```

8. Delete states on uninstall, as in */core/modules/comment/comment.install*

```
function comment_uninstall() {
  ...
  // Remove states.
  Drupal::state()->delete('comment.node_comment_statistics_scale');
}
```

9. Add test coverage to the upgrade tests, as in */core/modules/system/tests/upgrade/drupal-7.state.system.database.php*:

```
db_merge('variable')
  ->key(array('name' => 'node_cron_comments_scale'))
  ->fields(array('value' => serialize(1.0 / 1000)))
   ->execute();
```

Check if new values apply, as in */core/modules/system/lib/Drupal/system/Tests/Upgrade/StateSystemUpgradePathTest.php*:

```
$expected_state['comment.count_scale'] = array(
  'value' => 1.0 / 1000,
  'variable_name' => 'node_cron_comments_scale',
);
```

Chapter 5. Appendix: Style Guide Etc.

Formatting and Style Guide

The documentation in this project is formatted using AsciiDoc [http://asciidoc.org], which is a fairly simple markdown-style syntax. The AsciiDoc user guide [http://asciidoc.org/userguide.html] explains the markup syntax in detail. Or, you can use this handy AsciiDoc Cheat Sheet [http://powerman.name/doc/asciidoc].

A few notes on syntax that are specific to Drupal Core documentation follow.

Chapters and Sections

AsciiDoc supports a couple of different syntax options for chapters and sections; we're using an alternative that is not covered in all AsciiDoc documentation, but is simpler to use.

Also, every section and chapter should have an identifier on it.

So, it looks like this:

```
= Overall Book Title

[[first_chapter_id]]
== Chapter Title One

[[first_section_id]]
=== Section Title Sub 1
```

You can then make a link to a different chapter or section by putting

```
<<the_id>>
```

into your text.

Each chapter should be in a separate file; sections can be in the same file or separate files. Chapters are things like "Configuration API" and sections roughly correspond to drupal.org book pages within each chapter. On api.drupal.org, each section will be displayed on its own page.

Code

To include PHP code, prefix a literal block (which starts with `----` on its own line) with `[source,php]`. Other types of source code are also recognized, such as `css`, `javascript`, `sql`, `html`, etc. For generic source code, you can omit the `[source,php]` line. There are examples in most of the files.

Drupal Versions

This repository will be branched for new versions of Drupal. So, do not specifically use the version number in your writing, except in the top-level title, and in specific sections about updating from one version to another. The hope is that when we branch to the next version, we should just be able to delete the version-specific updating sections, change the main title in one place, and be done.

Literals and Text Formatting

File names and directories in text should be formatted in italics:

```
_core/modules/system/system.module_
```

Functions, variables, class names, snippets, etc. in text should be formatted in monospace:

```
+my_function_name()+
+$foo+
```

Comments

You can put comments about formatting into your AsciiDoc using double slashes, like PHP:

```
// This is a comment that is only relevant to someone reading
// the AsciiDoc source.
```