

Configuration and Settings

TABLE OF CONTENTS

- [Load some config](#)
- [Views](#)
- [Add config to an existing module](#)
- [Import something you changed in your module](#)
- [Config Storage in the database](#)
- [Add some config to site config form](#)
- [Override config in settings.php](#)
- [Setup a testing variable in config for a project](#)
- [Getting and setting configuration with drush](#)
- [Creating a module allowing users to edit/update some config](#)
- [Drush config commands](#)
 - [View config](#)
 - [Viewing overridden config values](#)
 - [Delete from config](#)
 - [Check what has changed with config:status](#)
 - [Export entire config](#)
 - [Import config changes](#)

views 129

Config is stored in yaml files so it can be checked into git. It is loaded into the config table of the database for performance. Use `drush config-import` (or `drush cim`) for this purpose. Config includes database table definitions, views definitions and lots more. You can even use config to store a little setting indicating your site is in a test mode which can trigger displaying some useful information that only you can see.

Config files should be stored in a non-web accessible directory and specified in settings.php e.g.

```
$settings['config_sync_directory'] = '../config/sync';
```

[More about Defining and using your own configuration in Drupal](#)

Load some config

This example shows how to load a rest endpoint from config. This is very similar to Drupal 7 `variable_get()`.

Use the Configuration API main entry point `\Drupal::config()` to load the config item and then use `get()` to retrieve the value you want. Config can have multiple values in a single yaml file.

```
$pizzaEndpoint = \Drupal::config('pizza_academy_core.pbx.rest.endpoint');  
$pizza_service_url = $pizzaEndpoint->get('pizza_rest_endpoint')->$reg_id;
```

When you export the config, this information is stored in a file called `pizza_academy_core.pbx.rest.endpoint.yml` with a key `pizza_rest_endpoint`.

The contents of the file are simply:

```
pizza_rest_endpoint: 'https://pbx.pizza.com/pbx-profile-service/'
```

You'll find this in a file in the config sync directory specified in settings.php e.g.

```
config/sync/pizza_academy_core.pbx.rest.endpoint.yml
```

The config sync directory location is specified in `settings.php` like this

```
$settings['config_sync_directory'] = '../config/sync';
```

[More on creating custom modules: Using your own configuration](#)

[Drupal::config API Reference](#)

You can override config items in a `settings.php` Or `local.settings.php` using the `$config` global variable.

Views

For views, the config filenames are be in the form `views.view.infofeeds` for a view called `infofeeds`.

Add config to an existing module

You simply create a yml file in the module's `/config/install` directory.

The config file should start with the module name then a period and the thing you want to store the config about. So `modulename.something.yml` e.g. `dir_salesforce.cron.yml` for cron information, `dir_funnelback.yml` for funnelback information or `tea_teks_spr.testing.yml` for testing information.

If the module name is `pizza_academy_core` and the thing I want to store config about is the `pbxpath`, I would create a file called `pizza_academy_core.pbxpath.yml`.

The yml filename is passed as a parameter into `\Drupal::config(...)` without the `.yml` extension. e.g. if the filename is `danamod.header_footer_settings.yml` then use:

```
$config = \Drupal::config('danamod.header_footer_settings');
```

Here we add some configuration to a module called `pizza_academy_core`:

In `docroot/modules/custom/pizza_academy_core/config/install/pizza_academy_core.pbxpath.yml`

We have a file with the contents:

```
url: 'https://pbx.pizza.com/'
langcode: 'en'
```

You can copy it to the `config/sync` directory, manually paste the contents into the config Drupal u/i or import it into the db with drush. The drush way is the easiest in my opinion.

```
drush config-import --source=modules/custom/pizza_academy_core/config/install/ --partial -y
```

Then you can access it from a controller at `docroot/modules/custom/pizza_academy_core/src/Controller/VerifyCertificationPage.php` using the following code:

```
$pbx_path_config = \Drupal::config('pizza_academy_core.pbxpath');
$pbx_path = $pbx_path_config->get('url');
$pbx_achievements_url = $pbx_path . "achievements?regid=" . $reg_id;
```

Once you grab the url, you can use it later in your code.

Import something you changed in your module

During module development, you might find you want to add some configuration. This is very useful as part of that workflow.

```
drush @dev2 config-import --source=modules/migrate/test1/config/install/ --partial -y
```

Note. the `@dev2` is a site alias. See [Drush alias docs for more info](#). These are sooo useful.

Config Storage in the database

Config is also kept in the config table of the database.

The name field stores the config id e.g. views.view.infofeeds (the definition of a view called infofeeds)

The data field stores the stuff in the config serialized into a blob

Add some config to site config form

Here we add a phone number to the site config. This is put in a .module file.

```
use Drupal\Core\Form\FormStateInterface;

/**
 * Implements hook_form_FORM_ID_alter().
 */
function mymodule_form_system_site_information_settings_alter(&$form, FormStateInterface $form_state) {

  $form['site_phone'] = [
    '#type' => 'tel',
    '#title' => t('Site phone'),
    '#default_value' =>
      Drupal::config('system.site')->get('phone'),
  ];

  $form['#submit'][] = 'mymodule_system_site_information_phone_submit';
}
```

The \$form['#submit'] modification adds our callback to the form's submit handlers. This allows our module to interact with the form once it has been submitted. The mymodule_system_site_information_phone_submit callback is passed the form array and form state. We load the current configuration factory to receive the configuration that can be edited. We then load system.site and save phone based on the value from the form state.

```
function mymodule_system_site_information_phone_submit(array &$form, FormStateInterface $form_state) {
  $config = Drupal::configFactory()->getEditable('system.site');
  $config->set('phone', $form_state->getValue('site_phone'))
    ->save();
}
```

Don't forget there is a [module called config pages](#) which might save you some coding if you need to add some config to a site.

Override config in settings.php

This can be useful for local development environment (where you might put these changes into settings.local.php) or on each one of your servers where you might need some configuration to be slightly different. e.g. dev/test/prod.

Drupal 9 allows global \$config overrides (similar to drupal 7) The configuration system integrates these override values via the Drupal\Core\Config\ConfigFactory::get() implementation. When you retrieve a value from configuration, the global \$config variable gets a chance to change the returned value:

```
// Get system site maintenance message text. This value may be overridden by
// default from global $config (as well as translations).
$message = \Drupal::config('system.maintenance')->get('message');
```

To override configuration values in global \$config in settings.php, use a line like this (which references the configuration keys:

```
$config['system.maintenance']['message'] = 'Sorry, our site is down now.';
```

For nested values, use nested array keys

```
$config['system.performance']['css']['preprocess'] = 0;
```

If you have a configuration change, for example, you have enabled google tag manager. When you export the config drush cex -y and git diff to see what changed in config, you'll see (in the last 2 lines) that status is changed from true to false.

```
$ git diff
```

```
diff --git a/config/sync/google_tag.container.default.yml b/config/sync/google_tag.container.default.yml
index 39e498c99..375bfb8af 100644
--- a/config/sync/google_tag.container.default.yml
+++ b/config/sync/google_tag.container.default.yml
@@ -1,6 +1,6 @@
uuid: 5919bbb9-95e3-4d8b-88c8-030e6a58ec6c
langcode: en
-status: false
+status: true
```

To put this in settings.php or settings.local.php, add a line and set the value to true or false:

```
$config['google_tag.container.default']['status'] = false;
```

Setup a testing variable in config for a project

First create the yml file in your module/config/install e.g. tea_teks_srp.testing.yml with this as the contents:

```
test_mode: FALSE
```

This will be the default state of the app

In the Drupal U/I under config, devel, configuration synchronization, import, single item i.e. at /admin/config/development/configuration/single/import select simple configuration. In the configuration name field, put tea_teks_srp.testing.

Paste in the text of the file

```
test_mode: FALSE
```

and import. This will load the new value into the database.

Then in your docroot/sites/default/settings.local.php (to enable testing features) add

```
$config['tea_teks_srp.testing']['test_mode'] = TRUE;
```

This will override your config you added above so test_mode is true.

Then to use the test_mode, you can load it into a controller class (or form class) from the config (and the value in the settings.local.php will override the default) with the following:

```
$test_mode = \Drupal::config('tea_teks_srp.testing')->get('test_mode');
```

And then just use the \$test_mode variable as needed e.g.

```
if ($this->test_mode) {
  $value = $this->t("Reject Citation $citation_nid");
}
```

Getting and setting configuration with drush

Here we are fiddling with the shield module settings

In config, synchronize, we see an item: shield.settings

So we can load it with drush:

```
$ drush cget shield.settings
credential_provider: shield
credentials:
  shield:
    user: nistor
    pass: blahblah
print: 'Please provide credentials for access.'
allow_cli: true
_core:
  default_config_hash: c1dcnGFTXFeMq2-Z8e7H6Qxp6TTJe-ZhSA126E3bQJ4
```

Drilling down deeper, let's say we want to view the credentials section. Notice that drush requires a space instead of a colon:

```
$ drush cget shield.settings credentials
'shield.settings:credentials':
  shield:
    user: nisor
    pass: blahblah
```

Now to get down to the user name and password. And we are adding period back in. Huh?

```
$ drush cget shield.settings credentials.shield
'shield.settings:credentials.shield':
  user: nisor
  pass: blahblah
```

and finally:

```
$ drush cget shield.settings credentials.shield.pass
'shield.settings:credentials.shield.pass': blahblah
```

So if you want to **set** these:

```
drush cset shield.settings credentials.shield.pass yomama

Do you want to update credentials.shield.pass key in shield.settings
config? (y/n): y
```

And

```
drush cset shield.settings credentials.shield.user fred

Do you want to update credentials.shield.pass key in shield.settings
config? (y/n): y
```

And there is that message

```
drush cget shield.settings print
'shield.settings:print': 'Please provide credentials for access.'
```

And so

```
drush cset -y shield.settings print "Credentials or I won't let you in"
```

and while we're here, we could always put these into the \$config object via settings.php (or settings.local.php :

```
$config['shield.settings']['credentials']['shield']['user'] = "nisor";
$config['shield.settings']['credentials']['shield']['pass'] = "blahblah";
```

Similarly, for setting stage_file_proxy origin:

```
drush config-set stage_file_proxy.settings
origin https://www.mudslinger.com
```

Creating a module allowing users to edit/update some config

When you want to add a form to allow the user to update the config, create a module with a form as you would anywhere else. The form will need the standard `buildForm()`, `submitForm()` and `getFormId()` methods.

e.g. in `docroot/modules/custom/danamod/src/Form/HeaderFooterForm.php`

In the `buildform`, you load the config object, then get each value from the object, load them into the form (in `#default_value` array items) so the user can see the current value.

```
// Load the values from config.
$config = \Drupal::config('danamod.header_footer_settings');

$address1 = $config->get('footer_address1');
$address2 = $config->get('footer_address2');
$address3 = $config->get('footer_address3');
$email = $config->get('footer_email');
$logo_url = $config->get('logo_url');

// And put them into the form render array.

$form['footer']['footer_address1'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Address line 1'),
  '#default_value' => $address1,
];
$form['footer']['footer_address2'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Address line 2'),
  '#default_value' => $address2,
];
```

In the `submitForm()` member, you extract the values from the `$form_state`, load an editable config object and `->set()` values and `->save()` them.

```
$config = \Drupal::configFactory()->getEditable('danamod.header_footer_settings');
$values = $form_state->getValues();
$address1 = $values['footer_address1'];
$address2 = $values['footer_address2'];
$config->set('footer_address1', $address1);
$config->set('footer_address2', $address2);

$config->save();

\Drupal::messenger()->addMessage('Values have been saved.');
```

And a little shorthand

```
$config
->set('display_stars', $display_stars)
->set('display_summary_summative', $display_summary_summative)
->save();

\Drupal::messenger()->addMessage("Values have been saved.");
```

Drush config commands

Drush will provide you with all the tools you need to fiddle with config from the command line. Check out the [drush docs](#)

View config

Note. when you view the value in config, drush cleverly will **ignore values** overridden in settings.php. More below.

cget is short for config:get.

From the [drush docs](#)

- drush config:get system.site - displays the system.site config.
- drush config:get system.site page.front - displays what Drupal is using for the front page of the site: e.g.

```
$ drush config:get system.site page.front
'system.site.page.front': /node
```

Viewing overridden config values

When you view the value in config, drush cleverly will **ignore values** overridden in settings.php.

- drush cget narcs_infoconnect.imagepath basepath

This displays the basepath that is in the Drupal database. If you override the basepath in settings.php, you have to use the special flag to see the overridden value.

```
drush cget narcs_infoconnect.imagepath basepath --include-overridden
```

Also drush can execute php for a little more fun approach:

```
drush ev "var_dump(\Drupal::configFactory()->getEditable('system.site')->get('name'))"
```

or

```
drush ev "print \Drupal::config('narcs_inferconnect.imagepath')->get('basepath');"
```

Delete from config

cdel is short for config:delete.

- drush \@dev2 cdel migrate_plus.migration.test1
- drush \@dev2 cdel migrate_plus.migration_group.default

Check what has changed with config:status

cst is short for config:status

```
drush cst
```

Name	State

admin_toolbar.settings	Only in DB
admin_toolbar_tools.settings	Only in DB
automated_cron.settings	Only in DB
block.block.bartik_account_menu	Only in DB
block.block.bartik_branding	Only in DB
block.block.bartik_breadcrumbs	Only in DB
block.block.bartik_content	Only in DB
...	

Only in DB means the config has not yet been exported. Best practice is to check the config info git for loading onto the production site. Usually you would use `drush cex` at this point to export the config and add it to git.

After exporting drush will report that everything has been exported and that there are no differences between the database and the sync folder.

```
drush cst
```

```
[notice] No differences between DB and sync directory.
```

Export entire config

`cex` is short for `config:export`. This will dump the entire config, a bunch of yml files into the config sync folder which is specified in `settings.php` (or `settings.local.php`) as

```
$settings['config_sync_directory'] = '../config/sync';
```

- `drush cex -y` - export entire config.

```
$ drush cex -y
```

```
[success] Configuration successfully exported to ../config/sync.  
../config/sync
```

Import config changes

If you change the site name (for example) by mistake and want to restore it, you can re-import the values from the last export.

First check what changed with `drush cst` then use `drush cim` to restore the config to its previous glory. `cim` is short for `config:import`.

Drupal cleverly notices which config items have changed and loads only those changes into the database.

```
$ drush cst
```

Name	State

system.site	Different

and


```
$ drush cim -y
```

```
+-----+-----+-----+
| Collection | Config   | Operation |
+-----+-----+-----+
|           | system.site | Update    |
+-----+-----+-----+
```



```
// Import the listed configuration changes?: yes.
```

```
[notice] Synchronized configuration: update system.site.
```

```
[notice] Finalizing configuration synchronization.
```

```
[success] The configuration was imported successfully.
```

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Politt](#) is licensed under [CC BY 4.0](#)  

Page last modified: Apr 13 2023.

[Edit this page on GitHub](#)

This site uses [Just the Docs](#), a documentation theme for Jekyll.