

[Skip to main content](#)

[Drupal Book](#)

- [Home](#)
- [Actions](#)
- [Attribution](#)
- [Batch and Queue](#)
- [Blocks](#)
- [Caching](#)
- [Composer](#)
- [Config](#)
- [Cron](#)
- [Dates and Times](#)
- [Debugging](#)
- [Development](#)
- [Drush](#)
- [Email](#)
- [Entities](#)
- [Forms](#)
- [General](#)
- [Getting off the Island](#)
- [Hooks](#)
- [Learning](#)
- [Links](#)
- [Logging](#)
- [Menus](#)
- [Migrate](#)
- [Modals](#)
- [Nodes and Fields](#)
- [Queries](#)
- [Redirects](#)
- [Render Arrays](#)
- [Routes](#)
- [Security](#)
- [Services](#)
- [State](#)
- [Taxonomy](#)
- [Tests](#)
- [Twig](#)
- [Fork me on GitHub](#)

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

- 
- 

Drupal at your fingertips

Drupal 9 & 10 developer's quick code reference

by Selwyn Polit

This book is a quick reference for developers creating Drupal sites. While working on such sites during my career, I have gathered a large collection of notes and code which I use for my reference. I hope it will be useful for you, too.

During my hunting through examples on [Drupal.org](#), [Stack Overflow](#) and many other places, I often wished there was a

succinct set of current annotated code samples explaining how to do things the [Drupal way](#). Of course, there is the excellent [examples module](#), which is very handy. But this book is designed to otherwise fill that gap. Please forgive the ~~todo~~ items, but I've found that if I want to get it perfect, it won't ever get published.

You are invited to contribute (in the open source way) by making edits and pull requests at [this link on Github](#). There is a very simple online editor which automatically forks the repo and lets you type your changes directly in your browser in [markdown format](#). Try it by editing a chapter of interest to you. And while you are here [see who else contributed to the masterpiece](#) 😊

If you are inspired, please be a part of this project by contributing your own thoughts, code snippets, tutorials, links etc. (Imposters are welcomed!) Add your name on the [attribution page](#). Either add a note in the comment form at the bottom of each page or click the Edit this page on GitHub also at the very bottom of any page and start editing. You don't need to know Git.

views 2538

- [Batch and Queue - Batch API](#)
- [Blocks - Generate the code to create blocks, dependency injection, blocks with config forms, block permissions](#)
- [Caching - Cache tags, disabling caching, caching REST resources, development setup, various caching tips](#)
- [Composer, updates and patches](#)
- [Config - Creating, loading, overriding, drush config commands](#)
- [Cron - Using hook_cron, sample crontab files, how to stop cron](#)
- [Dates - Getting and setting date fields, date arithmetic, expiration, node creation and changed dates, Smart date module](#)
- [Debugging with PhpStorm, DDEV and Xdebug](#)
- [Development - Local setup, creating sites, DDEV, troubleshooting, PHPStorm setup, Xdebug setup, Twig debugging](#)
- [Drush - the amazing command line tool that makes everyone's life so much better](#)
- [Email - Sending emails in Drupal](#)
- [Entities - More entities - the primary building blocks of Drupal](#)
- [Forms - Creating and modifying forms using the Form API, AJAX forms, modals](#)
- [General - Useful things that seemed to not fit well elsewhere](#)
- [Hooks - hooking into the Drupal hook system to make modifications](#)
- [Learning and keeping up with the latest on Drupal](#)
- [Links, Aliases and URLs](#)
- [Logging and notification](#)
- [Menus](#)
- [Migration - automated ways to get data into Drupal](#)
- [Modal dialogs](#)
- [Nodes and Fields - Loading and saving nodes and different types of fields, multivalue field magic, link fields, entity reference fields, url fields, storing JSON data into text fields, paragraphs, great cheat sheets](#)
- [Queries - entityQueries, SQL queries, inserting and deleting content](#)
- [Reaching out of Drupal](#)
- [Redirects](#)
- [Render Arrays](#)
- [Routes and Controllers](#)
- [Services and Dependency injection](#)
- [State API, Tempstore and UserData \(Interesting flavors of storage\)](#)
- [Taxonomy](#)
- [PHPUnit and Drupal Test Traits](#)
- [Twig - Formatting and controlling your output](#)

I'm working on writing more about:

- Events
 - Custom Plugins
 - Javascript
 - Media image files
 - Migration
 - Miscellaneous
 - Paragraphs
 - Security and Preventing attacks
 - Single sign on
 - Views
-

Page last modified: May 2 2023.

[Edit this page on GitHub](#)

Drupal at your fingertips



Drupal 9 & 10 developer's quick code reference

BY SELWYN POLIT

This book is a quick reference for developers creating Drupal sites. While working on such sites during my career, I have gathered a large collection of notes and code which I use for my reference. I hope it will be useful for you, too.

During my hunting through examples on [Drupal.org](#), [Stack Overflow](#) and many other places, I often wished there was a succinct set of current annotated code samples explaining how to do things the [Drupal way](#). Of course, there is the excellent [examples module](#), which is very handy. But this book is designed to otherwise fill that gap. Please forgive the `todo` items, but I've found that if I want to get it perfect, it won't ever get published.

You are invited to contribute (in the open source way) by making edits and pull requests at [this link on Github](#). There is a very simple online editor which automatically forks the repo and lets you type your changes directly in your browser in [markdown format](#). Try it by editing a chapter of interest to you. And while you are here, [see who else contributed to the masterpiece ☺](#)

If you are inspired, please be a part of this project by contributing your own thoughts, code snippets, tutorials, links etc. (Imposters are welcomed!) Add your name on the [attribution page](#). Either add a note in the comment form at the bottom of each page or click the [Edit this page](#) on GitHub also at the very bottom of any page and start editing. You don't need to know Git.

views 2539

- [Batch and Queue - Batch API](#)
- [Blocks - Generate the code to create blocks, dependency injection, blocks with config forms, block permissions](#)
- [Caching - Cache tags, disabling caching, caching REST resources, development setup, various caching tips](#)
- [Composer, updates and patches](#)
- [Config - Creating, loading, overriding, drush config commands](#)
- [Cron - Using hook_cron, sample crontab files, how to stop cron](#)
- [Dates - Getting and setting date fields, date arithmetic, expiration, node creation and changed dates, Smart date module](#)
- [Debugging with PhpStorm, DDEV and Xdebug](#)
- [Development - Local setup, creating sites, DDEV, troubleshooting, PHPStorm setup, Xdebug setup, Twig debugging](#)
- [Drush - the amazing command line tool that makes everyone's life so much better](#)

- [Email - Sending emails in Drupal](#)
 - [Entities - More entities - the primary building blocks of Drupal](#)
 - [Forms - Creating and modifying forms using the Form API, AJAX forms, modals](#)
 - [General - Useful things that seemed to not fit well elsewhere](#)
 - [Hooks - hooking into the Drupal hook system to make modifications](#)
 - [Learning and keeping up with the latest on Drupal](#)
 - [Links, Aliases and URLs](#)
 - [Logging and notification](#)
 - [Menus](#)
 - [Migration - automated ways to get data into Drupal](#)
 - [Modal dialogs](#)
 - [Nodes and Fields - Loading and saving nodes and different types of fields, multivalue field magic, link fields, entity reference fields, url fields, storing JSON data into text fields, paragraphs, great cheat sheets](#)
 - [Queries - entityQueries, SQL queries, inserting and deleting content](#)
 - [Reaching out of Drupal](#)
 - [Redirects](#)
 - [Render Arrays](#)
 - [Routes and Controllers](#)
 - [Services and Dependency injection](#)
 - [State API, Tempstore and UserData \(Interesting flavors of storage\)](#)
 - [Taxonomy](#)
 - [PHPUnit and Drupal Test Traits](#)
 - [Twig - Formatting and controlling your output](#)
-

I'm working on writing more about:

- Events
 - Custom Plugins
 - Javascript
 - Media image files
 - Migration
 - Miscellaneous
 - Paragraphs
 - Security and Preventing attacks
 - Single sign on
 - Views
-
-

[Back to top](#)

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

Page last modified: May 2 2023.

[Edit this page on GitHub](#)

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

Actions

TABLE OF CONTENTS

- [Overview](#)
- [Custom Actions](#)
- [Archive Node Action \(simple\)](#)
- [Update Node Title Custom Action Plugin with Configuration](#)
- [Reference](#)

views 292

Overview

The Actions module is a core module in Drupal 9 that allows site builders and developers to create automated workflows by defining actions and triggers. In Drupal, actions refer to a set of tasks or operations that can be performed on a site. For example, an action can be sending an email, publishing content, or updating a database record.

With the Actions module, you can create a customized workflow that can be triggered by a specific event. For example, when a user submits a form on your website, you can trigger an action to send an email notification to the site administrator. You can also chain multiple actions together to create complex workflows.

The Actions module provides a user-friendly interface for defining and managing actions. You can create custom actions using PHP code (see below) or use pre-defined actions provided by Drupal core or contributed modules. The module also provides a way to organize and categorize actions, making it easy to find and use them in your workflows.

TL;DR: The Actions module provides a powerful and flexible way to automate tasks on your website by defining actions and triggers, allowing you to create customized workflows that improve your site's functionality and user experience. - ChatGPT (with minor tweaks on my part.)

From the config page in Drupal:

There are two types of actions: simple and advanced. Simple actions do not require any additional configuration and are listed here automatically. Advanced actions need to be created and configured before they can be used because they have options that need to be specified; for example, sending an email to a specified address or unpublishing content containing certain words. To create an advanced action, select the action from the drop-down list in the advanced action section below and click the Create button.

NOTE

The [ECA module](#) can use Drupal core actions (and events) if you need to do even more.

Custom Actions

Here, A Drupal action is a functionality which performs specific action when executed. For example, Archive Node or Make Content Sticky.

Actions use the annotation class `Drupal\Core\Annotation\Action`, and extend `Drupal\Core\Action\ActionBase` OR `Drupal\Core\Action\ConfigurableActionBase` (if the action is configurable.)

Action Plugin definition is defined in Plugin Annotation. It has 3 required keys

```

/**
 * Provides an Archive Node Action.
 *
 * @Action(
 *   id = "action_plugin_examples_archive_node",
 *   label = @Translation("Archive Node"),
 *   type = "node",
 *   category = @Translation("Custom")
 * )
 */

```

id → ID of the Action Plugin

label → Name of the Action Plugin

type → Entity type to which the Action Plugin belongs to

category → (optional) Category of the Action Plugin

Archive Node Action (simple)

This is a simple action which requires no configuration. When it is run, it changes the alias of the node to /archive//. It also sets the title to have the word `Archive` in the front of it. Finally it disables the sticky and promoted flags. [It is also on gitlab here.]

(https://git.drupalcode.org/sandbox/Bhanu951-3103712/-/blob/8.x-dev/action_plugin_examples/src/Plugin/Action/ArchiveNode.php)

```

<?php

namespace Drupal\action_plugin_examples\Plugin\Action;

use Drupal\Core\Action\ActionBase;
use Drupal\Core\Session\AccountInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\pathauto\PathautoState;

/**
 * Provides an Archive Node Action.
 *
 * @Action(
 *   id = "action_plugin_examples_archive_node",
 *   label = @Translation("Archive Node"),
 *   type = "node",
 *   category = @Translation("Custom")
 * )
 */
class ArchiveNode extends ActionBase implements ContainerFactoryPluginInterface {

    /**
     * The Messenger service.
     *
     * @var \Drupal\Core\Messenger\MessengerInterface
     */
    protected $messenger;

    /**
     * Logger service.
     *
     * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
     */
}
```

```

    }

protected $logger;

/**
 * The path alias manager.
 *
 * @var \Drupal\path_alias\AliasManagerInterface
 */
protected $aliasManager;

/**
 * Language manager for retrieving the default Langcode.
 *
 * @var \Drupal\Core\Language\LanguageManagerInterface
 */
protected $languageManager;

/**
 * {@inheritDoc}
 */
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
  $instance = new static($configuration, $plugin_id, $plugin_definition);
  $instance->logger = $container->get('logger.factory')->get('action_plugin_examples');
  $instance->messenger = $container->get('messenger');
  $instance->aliasManager = $container->get('path_alias.manager');
  $instance->languageManager = $container->get('language_manager');
  return $instance;
}

/**
 * {@inheritDoc}
 */
public function access(NodeInterface $node, AccountInterface $account = NULL, $return_as_object = FALSE) {
  /** @var \Drupal\node\NodeInterface $node */
  $access = $node->access('update', $account, TRUE)
    ->andIf($node->title->access('edit', $account, TRUE));
  return $return_as_object ? $access : $access->isAllowed();
}

/**
 * {@inheritDoc}
 */
public function execute(NodeInterface $node = NULL) {

  /** @var \Drupal\node\NodeInterface $node */

  $language = $this->languageManager->getCurrentLanguage()->getId();

  $old_alias = $this->aliasManager->getAliasByPath('/node/' . $node->id(), $language);

  $title = $node->getTitle();
  $date = $node->created->value;
  $year = date('Y', $date);
  // $old_alias = $node->path->alias;
  $new_title = $this->t(['[Archive] | @title', '@title' => $title]);
  $node->setTitle($new_title);
  $node->setSticky(FALSE);
  $node->setPromoted(FALSE);

  $new_alias = '/archive/' . $year . $old_alias;
}

```

```

$node->set("path", [
  'alias' => $new_alias,
  'langcode' => $language,
  'pathauto' => PathautoState::SKIP,
]);
}

$node->save();

$message = $this->t('Node with NID : @id Archived.', ['@id' => $node->id()]);

$this->logger->notice($message);
$this->messenger->addMessage($message);

}
}

```

In order to get Action Plugin Discoverable you need to add `system.action.<plugin_id>.yml` which is placed in config/install

The structure of the .yml file is shown below:

```

langcode: en
status: true
id: action_plugin_examples_archive_node
label: 'Archive Node'
type: node
plugin: action_plugin_examples_archive_node

```

Created Action Plugin can be viewed on the `/admin/content` page.

The screenshot shows the Drupal administration interface for content management. At the top, there's a search bar and several filter options: Title, Content type, Published status, and Language, each with a dropdown menu. Below the filters is a 'Filter' button. The main area displays a table of content items. A context menu is open over one of the rows, with 'Archive Node' selected and highlighted in blue. Other options in the menu include: Update Node Title, Delete content, Make content sticky, Make content unsticky, Promote content to front page, Publish content, Save content, Remove content from front page, and Unpublish content. The table below has columns for Content type, Author, Status, Updated, and Operations. One row is visible, showing 'Article' as the content type, 'admin' as the author, 'Published' as the status, '05/10/2023 - 06:48' as the updated time, and 'Edit' and 'Delete' buttons in the operations column.

Update Node Title Custom Action Plugin with Configuration

This example updates a node title. It gets the new title info from configuration. [It is also here on Gitlab.](#)

```

<?php

namespace Drupal\action_plugin_examples\Plugin\Action;

use Drupal\Core\Action\ConfigurableActionBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Session\AccountInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
*/

```

```

* Provides an Update Node Title Action.
*
* @Action(
*   id = "action_plugin_examples_update_node_title",
*   label = @Translation("Update Node title"),
*   type = "node",
*   category = @Translation("Custom")
* )
*/
class UpdateNodeTitle extends ConfigurableActionBase implements ContainerPluginInterface {

    /**
     * The Messenger service.
     *
     * @var \Drupal\Core\Messenger\MessengerInterface
     */
    protected $messenger;

    /**
     * Logger service.
     *
     * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
     */
    protected $logger;

    /**
     * The path alias manager.
     *
     * @var \Drupal\path_alias\AliasManagerInterface
     */
    protected $aliasManager;

    /**
     * Language manager for retrieving the default Langcode.
     *
     * @var \Drupal\Core\Language\LanguageManagerInterface
     */
    protected $languageManager;

    /**
     * {@inheritDoc}
     */
    public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
        $instance = new static($configuration, $plugin_id, $plugin_definition);
        $instance->logger = $container->get('logger.factory')->get('action_plugin_examples');
        $instance->messenger = $container->get('messenger');
        $instance->aliasManager = $container->get('path_alias.manager');
        $instance->languageManager = $container->get('language_manager');
        return $instance;
    }

    /**
     * {@inheritDoc}
     */
    public function defaultConfiguration() {
        return ['title' => 'Updated New Title'];
    }

    /**
     * {@inheritDoc}
     */

```

```

/*
public function buildConfigurationForm(array $form, FormStateInterface $form_state) {
  $form['title'] = [
    '#title' => $this->t('New Title'),
    '#type' => 'textfield',
    '#required' => TRUE,
    '#default_value' => $this->configuration['title'],
  ];
  return $form;
}

/** 
 * {@inheritDoc}
 */
public function submitConfigurationForm(array &$form, FormStateInterface $form_state) {
  $this->configuration['title'] = $form_state->getValue('title');
}

/** 
 * {@inheritDoc}
 */
public function access($node, AccountInterface $account = NULL, $return_as_object = FALSE) {
  /** @var \Drupal\node\NodeInterface $node */
  $access = $node->access('update', $account, TRUE)
    ->andIf($node->title->access('edit', $account, TRUE));
  return $return_as_object ? $access : $access->isAllowed();
}

/** 
 * {@inheritDoc}
 */
public function execute($node = NULL) {
  /** @var \Drupal\node\NodeInterface $node */
  $old_title = $node->getTitle();
  $updated_title = $this->t('@new_title @old_title', ['@old_title' => $old_title, '@new_title' => $this->configuration['title']]);
  $node->setTitle($updated_title)->save();

  $message = $this->t('Node Title for Node with NID : @id Updated.', ['@id' => $node->id()]);
  $this->logger->notice($message);
  $this->messenger->addMessage($message);
}

}

```

The Schema for Action Plugin `action_plugin_examples.schema.yml` which is in `config/schema`:

```

action_plugin_examples.configuration.action_plugin_examples_update_node_title:
  type: mapping
  label: 'Configuration for "Update node title" action'
  mapping:
    title:
      type: string
      label: Title

```

Action Plugin Definition `system.action.<plugin_id>.yml` is in `config/install`:

```
langcode: en
status: true
dependencies:
  module:
    - node
    - pathauto
id: action_plugin_examples_update_node_title
label: 'Update Node Title'
type: node
plugin: action_plugin_examples_update_node_title
configuration:
  title: 'Updated New Title'
```

Note. You need to install the core module Actions first to be able to configure action plugins in UI.

Configuration Page </admin/config/system/actions>

On the top of the page your new plugin should be present in the select list for creating advanced actions. When you select the plugin you should see the configuration form of your plugin:

See the [change record: Actions are now plugins, configured actions are configuration entities](#)

Reference

- [Actions UI module overview updated January 2023](#)
 - [Change Record - Actions are now plugins, configured actions are configuration entities](#)
 - [ECA: Event-Condition-Action - no-code solution to orchestrate your Drupal site](#)
-
-

[Back to top](#)

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

Page last modified: May 13 2023.

[Edit this page on GitHub](#)

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

Attribution

views 124

Thanks to these rockstars for their contribution to the project. Please add your name below!

[Danny Lufkin](#) - provided inspiration and hopefully some front-end code.

[Phillip Norton](#) - provided editing and a lot of material from his excellent articles at <https://www.hashbangcode.com>

[Karoly Negyesi](#) - provided some useful edits and corrections.

[Arek Susecki](#) Fixes to development chapter.

[Nic Laflin](#) Form section suggestions.

[Ales Rebec](#) - Styling help and email chapter.

[Sean Adams-Hiett](#) - Fixes.

[Gary Gao](#) - Query access checking.

[Maks Oleksyuk](#) - Amazing work on moving the book to a new theme and many other items.

[Bernard Martinez](#) - Block chapter fixes.

[Bronius Motekaitis](#) - Migration chapter.

[DarkDim](#) - Learning chapter.

[Balint Csuthy](#) - Fix link

[Bhanu](#) - Actions chapter and a bunch of other interesting issues

[John Picozzi](#) - Bringing us more in line with Drupal standards.

[Tony G.](#) - Help with typos

[James Shields](#) - Custom favicon.

[Manas Khatua](#) - Fix typos

[Back to top](#)

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

Page last modified: May 9 2023.

[Edit this page on GitHub](#)

Batch Processing and the Drupal Queue System

TABLE OF CONTENTS

- [Batch Processing Using the Batch API](#)
 - [Overview](#)
 - [Using the Batch API with a form](#)
 - [Using the Batch API from a controller](#)
 - [Using the Batch API with hook_update](#)
 - [Important rules about functions when using Batch API](#)
 - [Looking at the source](#)
 - [Passing parameters to the functions in a batch operation](#)
- [Queue System](#)
- [Resources](#)

views 375

Batch Processing Using the Batch API

Overview

The Batch API provides very useful functionality that lets you do work by breaking it into pieces to avoid PHP timeouts, etc. Usually, you'll do this by creating a group of node id's (using `array_chunk`) to be processed and use the batch API to process those arrays (chunks) of ids. You can also provide code that will let it figure out how much work to do and stop itself.

In addition, you create a function to handle things once all the chunks are complete. You can also give the Batch API a bunch of work to do and have it figure out for itself when it is finished.

The Batch API uses the Drupal Queue system, allowing it to pick up where it left off in case of problems.

You can use the Batch API in controllers, forms, hook updates, and Drush commands. The implementation of each one is slightly different, as you can see in the examples.

Most often you start a batch from a form where you fill in some options and click a button. You can also run a batch on a controller where the batch operation starts when you point the browser at a URL. Batch operations in Drush commands are started when you type the command in the terminal.

In the code: For a drush command, you use `drush_backend_batch_process()` to kick off the batch (from `DirtSalesforceController.php`):

```
if ($browser === FALSE) {
  drush_backend_batch_process();
}
```

Here is an explanation from the source of what it does:

```

 * Process a Drupal batch by spawning multiple Drush processes.
 *
 * This function will include the correct batch engine for the current
 * major version of Drupal, and will make use of the drush_backend_invoke
 * system to spawn multiple worker threads to handle the processing of
 * the current batch, while keeping track of available memory.
 *
 * The batch system will process as many batch sets as possible until
 * the entire batch has been completed or 60% of the available memory
 * has been used.
 *
 * This function is a drop in replacement for the existing batch_process()
 * function of Drupal.

```

Although it will be covered in more detail in the future, it is useful to know that there is now a batch builder: The [BatchBuilder class](#) for batch API. It provides a more streamlined object-oriented approach to creating batches.

Using the Batch API with a form

This example replaces a multivalue field with some new values processing 10 nodes at a time. The decision to process 10 at a time is arbitrary, but be aware that the more nodes you process at a time, the higher the possibility that the process will time out causing the batch will fail.

The form example is in the accompanying source and is accessed at <https://d9book.ddev.site/batch-examples/batchform>

The source file is at `web/modules/custom/batch_examples/src/Form/BatchForm.php` and is presented in pieces below:

Here is a simple form with a button used to kick off the batch operation.

```

/**
 * {@inheritDoc}
 */
public function buildForm(array $form, FormStateInterface $form_state): array {
  $form['message'] = [
    '#markup' => $this->t('Click the button below to kick off the batch!'),
  ];
  $form['actions'][#type] = 'actions';
  $form['actions'][submit] = [
    '#type' => 'submit',
    '#value' => $this->t('Run Batch'),
  ];
}

return $form;
}

```

The `submitForm()` method calls `updateEventPresenters()`.

```

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state): void {
  $this->updateEventPresenters();
  $this->messenger()->addStatus($this->t('The message has been sent.'));
  $form_state->setRedirect('<front>');
}

```

The route is:

```

batch_examples.batch:
path: '/batch-examples/batchform'
defaults:
  _title: 'Batch Form'
  _form: 'Drupal\batch_examples\Form\BatchForm'
requirements:
  _permission: 'access content'

```

Here is the `updateEventPresenters()` method. Notice the `$operations` array, which contains the function to call to do the work of each batch as well as the list of nids to process.

```

function updateEventPresenters(): void {
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'event')
    ->sort('title', 'ASC')
    ->accessCheck(TRUE);
  $nids = $query->execute();

  // Create batches.
  $chunk_size = 10;
  $chunks = array_chunk($nids, $chunk_size);
  $num_chunks = count($chunks);

  // Submit batches.
  $operations = [];
  for ($batch_id = 0; $batch_id < $num_chunks; $batch_id++) {
    $operations[] = [
      '\Drupal\batch_examples\Form\BatchForm::exampleProcessBatch',
      [
        $batch_id + 1,
        $chunks[$batch_id]
      ]
    ];
  }
  $batch = [
    'title' => $this->t('Updating Presenters'),
    'init_message' => $this->t('Starting to process events.'),
    'progress_message' => $this->t('Completed @current out of @total batches.'),
    'finished' => '\Drupal\batch_examples\Form\BatchForm::batchFinished',
    'error_message' => $this->t('Event processing has encountered an error.'),
    'operations' => $operations,
  ];
  batch_set($batch);
}

```

Here is the method that actually does the work. Most of the code is for information reporting. The actual work is in the `foreach $nids as $nid` loop:

```

public static function exampleProcessBatch(int $batch_id, array $nids, array &$context): void {
  if (!isset($context['sandbox'])['progress'])) {
    $context['sandbox']['progress'] = 0;
    $context['sandbox']['current_node'] = 0;
    $context['sandbox']['max'] = 0;
  }
  if (!isset($context['results'])['updated'])) {
    $context['results']['updated'] = 0;
    $context['results']['skipped'] = 0;
    $context['results']['failed'] = 0;
    $context['results']['progress'] = 0;
  }

  // Keep track of progress.
  $context['results']['progress'] += count($nids);
  $context['results']['process'] = 'Import request files';
  // Message above progress bar.
  $context['message'] = t('Processing batch #@batch_id batch size @batch_size for total @count items.', [
    '@batch_id' => number_format($batch_id),
    '@batch_size' => number_format(count($nids)),
    '@count' => number_format($context['sandbox']['max']),
  ]);

  foreach ($nids as $nid) {
    $filename = '';
    /** @var \Drupal\node\NodeInterface $event_node */
    $event_node = Node::load($nid);
    if ($event_node) {
      $array = ['Mary Smith', 'Fred Blue', 'Elizabeth Queen'];
      shuffle($array);
      $event_node->field_presenter = $array;
      $event_node->save();
    }
  }
}

```

The Form API will take care of getting the batches executed. If you aren't using a form, you use `batch_process()` like the line shown below. For this method, you specify any valid alias and the system will redirect to that alias after the batch completes.

```

// node/1 should be a valid node.
return batch_process('node/1');

```

NOTE

Also you can set up a `$batch` array with a title and a progress message with some variables that will get displayed.

You specify a `finished` index, which identifies a function to call after the batch is finished processing, as in the example below.

```
'finished' => '\Drupal\batch_examples\Form\BatchForm::batchFinished',
```

Here is the `batchFinished()` method, which displays and logs the results.

```

use Symfony\Component\HttpFoundation\RedirectResponse;

/**
 * Handle batch completion.
 *
 * @param bool $success
 *   TRUE if all batch API tasks were completed successfully.
 * @param array $results
 *   An array of processed node IDs.
 * @param array $operations
 *   A list of the operations that had not been completed.
 * @param string $elapsed
 *   Batch.inc kindly provides the elapsed processing time in seconds.
 */
public static function batchFinished(bool $success, array $results, array $operations, string $elapsed): RedirectResponse {
  $messenger = \Drupal::messenger();
  if ($success) {
    $messenger->addMessage(t('@process processed @count nodes, skipped @skipped, updated @updated, failed @failed in @elapsed.', [
      '@process' => $results['process'],
      '@count' => $results['progress'],
      '@skipped' => $results['skipped'],
      '@updated' => $results['updated'],
      '@failed' => $results['failed'],
      '@elapsed' => $elapsed,
    ]));
    \Drupal::logger('d9book')->info(
      '@process processed @count nodes, skipped @skipped, updated @updated, failed @failed in @elapsed.', [
        '@process' => $results['process'],
        '@count' => $results['progress'],
        '@skipped' => $results['skipped'],
        '@updated' => $results['updated'],
        '@failed' => $results['failed'],
        '@elapsed' => $elapsed,
      ]);
  }
  else {
    // An error occurred.
    // $operations contains the operations that remained unprocessed.
    $error_operation = reset($operations);
    $message = t('An error occurred while processing %error_operation with arguments: @arguments', [
      '%error_operation' => $error_operation[0],
      '@arguments' => print_r($error_operation[1], TRUE),
    ]);
    $messenger->addError($message);
  }
  // Optionally redirect back to the form.
  return new RedirectResponse('/batch-examples/batchform');
}

```

Using the Batch API from a controller

The Batch API is often used in connection with forms. If you're using a page callback, you will need to setup all the items, submit them to the batch API, and then call `batch_process()` with a url as the argument.

```
return batch_process('node/1');
```

After the batch is complete, Drupal will redirect you to that url. E.g. /node/1

[SEE ALSO](#)

In this example of a processing function, you can see error handling, logging, and tracking while retrieving files from a remote source. This is fairly common when moving data between systems. The rest of the code is almost identical to the previous example.

```
public static function fileImportProcessBatch(int $batch_id, array $nids, array &$context): void {
  if (!isset($context['sandbox'])['progress'])) {
    $context['sandbox']['progress'] = 0;
    $context['sandbox']['current_node'] = 0;
    $context['sandbox']['max'] = 0;
  }
  if (!isset($context['results'])['updated'])) {
    $context['results']['updated'] = 0;
    $context['results']['skipped'] = 0;
    $context['results']['failed'] = 0;
    $context['results']['progress'] = 0;
  }
  // Total records to process for all batches.
  if (empty($context['sandbox']['max'])) {
    $query = \Drupal::entityQuery('node')
      ->condition('status', 1)
      ->condition('type', 'opinion_request');
    $total_nids = $query->execute();
    $context['sandbox']['max'] = count($total_nids);
  }

  // Keep track of progress.
  $context['results']['progress'] += count($nids);
  $context['results']['process'] = 'Import request files';
  // Message above progress bar.
  $context['message'] = t('Processing batch #@batch_id batch size @batch_size for total @count items.', [
    '@batch_id' => number_format($batch_id),
    '@batch_size' => number_format(count($nids)),
    '@count' => number_format($context['sandbox']['max']),
  ]);

  $fileRepository = \Drupal::service('file.repository');
  foreach ($nids as $nid) {
    $filename = '';
    $request_node = Node::load($nid);
    if ($request_node) {
      $file_id = $request_node->get('field_request_file')->target_id;
      if (!empty($file_id)) {
        // confirm that the file exists.
        $file = File::load($file_id);
        if ($file) {
          $uri = $file->getFileUri();
          if (file_exists($uri)) {
            $context['results']['skipped']++;
            continue;
          }
        }
      }
    }
    // Skip retrieving file if there is no request date.
    $request_date = $request_node->get('field_request_date')->value;
    if (empty($request_date)) {
      $context['results']['skipped']++;
      continue;
    }
  }
}
```

```

\$source_url = $request_node->get('field_pdf_file_with_path')->value;
if (\$source_url == "missing") {
  \$context['results']['skipped]++;
  continue;
}
if (empty(\$source_url)) {
  \$filename = basename(\$source_url);
  if (empty(\$filename)) {
    \Drupal::logger('oag_opinions')
      ->error('file_import - Error retrieving file - invalid filename in' . \$source_url);
    \$context['results']['skipped]++;
    continue;
  }
  \$file_contents = @file_get_contents(\$source_url);
  if (\$file_contents === FALSE) {
    \Drupal::logger('oag_opinions')
      ->error('file_import - Error retrieving file ' . \$source_url);
    \Drupal::messenger()->addError(t('Error retrieving file %filename.', ['%filename' => \$source_url]), FALSE);
    \$context['results']['failed]++;
    continue;
  }
  \$destination = "public://" . \$filename;
  \$file = null;
  try {
    \$file = \$fileRepository->writeData(\$file_contents, \$destination, FileSystemInterface::EXISTS_REPLACE);
  }
  catch (FileNotFoundException \$e) {
    \Drupal::logger('oag_opinions')->error('file_import - Error saving file ' . \$destination);
    \Drupal::messenger()->addError(t('Error saving file %filename', ['%filename' => \$destination]), FALSE);
    \$context['results']['failed]++;
    continue;
  }
  if (!\$file) {
    \$context['results']['failed]++;
    continue;
  }
  \$fid = \$file->id();
  \$request_node->set('field_request_file', \$fid);
  //\$request_node->field_request_file->target_id = \$fid;
  \$request_node->save();
  \$context['results']['updated]++;
}
}

```

Here is the code that creates the batches and submits them.

```
public function summaryImport() {  
    $this->summaryCreateBatches();  
    return batch_process('/admin/content');  
}
```

Using the Batch API with hook_update

If you want to update the default value of a field for all nodes using the Batch API and hook_update_N checkout the following links:

- Using the Batch API and hook_update_N in Drupal 8

- [Drupal API | batch_example_update_8001 | batch_example.install](#)

Important rules about functions when using Batch API

All batch functions must be `public static` functions and all functions calling those must be explicitly namespaced like:

```
$nid = \Drupal\dirt_salesforce\Controller\DirSalesforceController::lookupCommodityItem($commodity_item_id);
```

You can't use `$this->my_function` even if they are in the same class. Grab the namespace from the top of the PHP file you are using. In this case:

```
namespace Drupal\dirt_salesforce\Controller;
```

You can however refer to the functions with `self::` e.g.

```
$node_to_update_dirt_contact_nid = self::getFirstRef($node_to_update, 'field_sf_dirt_contact_ref');
```

Looking at the source

The [source code for the Batch API](#) is really well commented and worth reading.

PASSING PARAMETERS TO THE FUNCTIONS IN A BATCH OPERATION

In this file <https://git.drupalcode.org/project/drupal/-/blob/10.1.x/core/includes/form.inc#L562-678>, there is an example batch that defines two operations that call `my_function_1` and `my_function_2`. Notice how parameters can be passed to `my_function_1` separated by commas. From <https://git.drupalcode.org/project/drupal/-/blob/8.7.8/core/includes/form.inc#L570>:

```
* Example:
* @code
*$batch = array(
*   'title' => t('Exporting'),
*   'operations' => array(
*     array('my_function_1', array($account->id(), 'story')),
*     array('my_function_2', array()),
*   ),
*   'finished' => 'my_finished_callback',
*   'file' => 'path_to_file_containing_my_functions',
* );
* batch_set($batch);
* // Only needed if not inside a form_submit handler.
* // Setting redirect in batch_process.
* batch_process('node/1');
```

NOTE

To execute the batch, the example shows a call to `batch_process('node/1')`. This could be any valid url alias e.g., `/admin/content`.

So here are the arguments for `my_function_1`:

```
* function my_function_1($uid, $type, &$context) {
```

You call the batch finished function with the following arguments:

```

/**
 * Handle batch completion.
 *
 * @param bool $success
 *   TRUE if all batch API tasks were completed successfully.
 * @param array $results
 *   An array of processed node IDs. - or whatever you put in $context['results'][]
 * @param array $operations
 *   A list of the operations that had NOT been completed.
 * @param $elapsed
 *   Elapsed time for processing in seconds.
 */
public static function batchFinished($success, array $results, array $operations, $elapsed) {

```

The results are displayed:

```

$messenger = \Drupal::messenger();
if ($success) {
  $messenger->addMessage(t('Processed @count nodes in @elapsed.', [
    '@count' => count($results),
    '@elapsed' => $elapsed,
  ]));
}

```

You can load the \$results array with all sorts of interesting data, such as:

```

$context['results']['skipped'] = $skipped;
$context['results']['updated'] = $updated;

```

Batch API provides a nice way to display detailed results using code like:

```

$messenger->addMessage(t('Processed @count nodes, skipped @skipped, updated @updated in @elapsed.', [
  '@count' => $results['nodes'],
  '@skipped' => $results['skipped'],
  '@updated' => $results['updated'],
  '@elapsed' => $elapsed,
]));

```

Which produce the following output:

```
Processed 50 nodes, skipped 45, updated 5 in 3 sec.
```

You can display an informative message above the progress bar this way.

I filled in the \$context['sandbox']['max'] with a value, but I could have used \$context['sandbox']['whole-bunch'] or any variable here.

```
$context['sandbox']['max'] = count($max_nids);
```

An informative message above the progress bar using number_format puts commas in the number if it is over 1,000.

```

$context['message'] = t('Processing total @count nodes',
  ['@count' => number_format($context['sandbox']['max'])]
);

```

Also you could show something about which batch number is running.

```

$operation_details = 'Yoyoma';
$id = 9;
$context['message'] = t('Running Batch "@id" @details',
  ['@id' => $id, '@details' => $operation_details]
);

```

You do have to provide your own info for the variables.

You can also stop the batch engine yourself with something like this. If you don't know beforehand how many records you need to process, you could use code like this.

```

// Inform the batch engine that we are not finished,
// and provide an estimation of the completion level we reached.
if ($context['sandbox']['progress'] != $context['sandbox']['max']) {
  $context['finished'] = ($context['sandbox']['progress'] >= $context['sandbox']['max']);
}

```

Queue System

From [Alan Saunders article](#) on December 2021:

A queue is simply a list of stuff that gets worked through one by one, one analogy could be a conveyor belt on a till in a supermarket, the cashier works through each item on the belt to scan them.

Queues are handy in Drupal for chunking up large operations, like sending emails to many people. By using a queue, you are trying to avoid overloading the servers resources which could cause the site to go offline until the resources on the server are free'd up.

From [Sarthak TTN](#) on Feb 2017:

This is the submitForm() which creates an item and puts it in the queue.

```

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state): void {
  /** @var QueueFactory $queue_factory */
  $queue_factory = \Drupal::service('queue');
  /** @var QueueInterface $queue */
  $queue = $queue_factory->get('email_processor');
  $item = new \stdClass();
  $item->username = $form_state->getValue('name');
  $item->email = $form_state->getValue('email');
  $item->query = $form_state->getValue('query');
  $queue->createItem($item);
}

```

Then you create a Queue Worker that implements ContainerFactoryPluginInterface and in the processItem() it processes a single item from the queue.

```

namespace Drupal\my_module\Plugin\QueueWorker;

use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Queue\QueueWorkerBase;
use Drupal\Core\Mail\MailManager;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * {@inheritDoc}
 */
class EmailEventBase extends QueueWorkerBase implements ContainerFactoryPluginInterface {

  /**
   * @param Drupal\Core\Mail\MailManager $mail
   *   The mail manager.
   */
  public function __construct(protected MailManager $mail) {}

  /**
   * {@inheritDoc}
   */
  public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    return new static($container->get('plugin.manager.mail'));
  }

  /**
   * Processes a single item of Queue.
   */
  public function processItem($data) {
    $params['subject'] = t('query');
    $params['message'] = $data->query;
    $params['from'] = $data->email;
    $params['username'] = $data->username;
    $to = \Drupal::config('system.site')->get('mail');
    $this->mail->mail('my_module', 'query_mail', $to, 'en', $params, NULL, true);
  }
}

```

Then you'll need a cronEventProcessor which in annotation tells cron how often to run the job:

```

namespace Drupal\my_module\Plugin\QueueWorker;

/**
 *
 * @QueueWorker(
 *   id = "email_processor",
 *   title = "My custom Queue Worker",
 *   cron = {"time" = 10}
 * )
 */
class CronEventProcessor extends EmailEventBase {}

```

Resources

Read more about batch processing at these sites:

- [Smack My Batch Up : Batch Processing In Drupal 8](#) by Phil Norton July 2016
- Highly commented [source code for batch operations around line 561 for Drupal 10](#) (or search for 'batch operations')

Read more about the Queue API at these sites:

- Karim Boudjema from August 2018 has [some good examples using the queue API](#)
 - Sarthak TTN from Feb 2017 shows some [sample code on implementing cron and the queue API](#)
 - [There is a somewhat incomplete example](#) From Alan Saunders article on December 2021
-
-

[Back to top](#)

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

Page last modified: Jul 29 2023.

[Edit this page on GitHub](#)

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

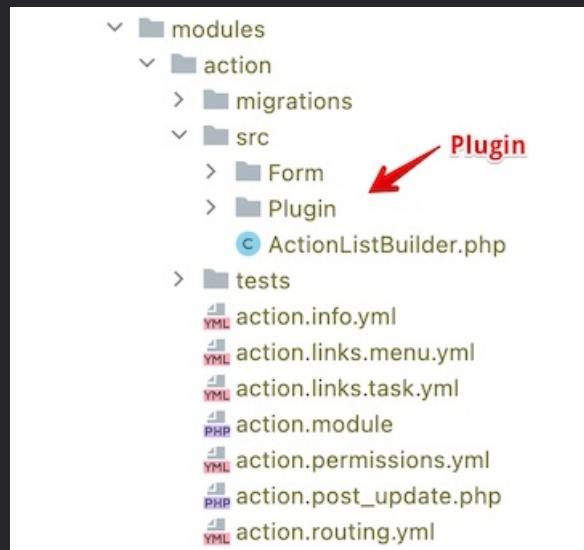
Blocks

TABLE OF CONTENTS

- [Create a block with Drush generate](#)
- [Anatomy of a custom block with dependency injection](#)
- [Create a block with an entityQuery](#)
- [Create a Block with a corresponding config form](#)
 - [The config form definition](#)
 - [The routing.yml file](#)
 - [The Block definition](#)
- [Modify a block with hook_block_view_alter or hook_block_build_alter](#)
- [Disable caching in a block](#)
- [Add a configuration form to your block](#)
- [Block display not updating after changing block content](#)
- [Block Permission \(blockAccess\)](#)
 - [Blocks shouldn't talk to the router, NodeRouteContext and friends should](#)
 - [Values returned by blockAccess\(\)](#)

views 173

Blocks are plugins, which are reusable pieces of code following design patterns. Plugins are also used to define views arguments, field formatters, field widgets, etc. The source files for blocks are found in each module's /src/Plugin directory.



SEE ALSO

- [Plugin API overview](#)
- [Annotations-based plugins](#)

Create a block with Drush generate

Use Drush's code generation ability to quickly generate the code you need to create your own custom block.

First generate a module if you don't have one. Here we generate a module called Block Module with a machine name: block_module.

\$ drush generate module

Welcome to module generator!

Module name \[Web\]:

► Block Module

Module machine name \[block_module\]:

►

Module description \[Provides additional functionality for the site.\]:

► Custom module to explore Drupal blocks

Package \[Custom\]:

►

Dependencies (comma separated):

►

Would you like to create module file? \[No\]:

► yes

Would you like to create install file? \[No\]:

►

Would you like to create libraries.yml file? \[No\]:

►

Would you like to create permissions.yml file? \[No\]:

►

Would you like to create event subscriber? \[No\]:

►

Would you like to create block plugin? \[No\]:

►

Would you like to create a controller? \[No\]:

►

Would you like to create settings form? \[No\]:

►

The following directories and files have been created or updated:

```
•  
/Users/selwyn/Sites/ddev93/web/modules/custom/block_module/block_module.info.yml  
  
•  
/Users/selwyn/Sites/ddev93/web/modules/custom/block_module/block_module.module
```

Use drush generate to create the code for a block. Specify the module name (e.g. block_module) so Drush knows where to put the block code. We also must give the block an admin label, plugin ID, and class.

```
$ drush generate block
```

Welcome to block generator!

Module machine name \[web]:

► block_module

Block admin label \[Example]:

► Block Module Example

Plugin ID \[block_module_block_module_example]:

►

Plugin class \[BlockModuleExampleBlock]:

►

Block category \[Custom]:

►

Make the block configurable? \[No]:

►

Would you like to inject dependencies? \[No]:

►

Create access callback? \[No]:

►

The following directories and files have been created or updated:

```
•  
/Users/selwyn/Sites/ddev93/web/modules/block_module/src/Plugin/Block/BlockModuleExampleBlock.php
```

This generates a file at web/modules/custom/block_module/src/Plugin/Block/BlockModuleExampleBlock.php that looks like this:

```
<?php

namespace Drupal\block_module\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/**
 * Provides a block module example block.
 *
 * @Block(
 *   id = "block_module_block_module_example",
 *   admin_label = @Translation("Block Module Example"),
 *   category = @Translation("Custom")
 * )
 */
class BlockModuleExampleBlock extends BlockBase {

  /**
   * {@inheritDoc}
   */
  public function build() {
    $build['content'] = [
      '#markup' => $this->t('It works!'),
    ];
    return $build;
  }
}
```

Enable the module with:

```
ddev drush en block_module
```

clear the cache with:

```
ddev drush cr
```

In Drupal, navigate to /admin/structure/block and place the block ("block module example") in the content area. See the diagram below on how to place the block in the content area.

Block layout | Drush Site-Install x +

Back to site Manage Shortcuts admin

Content Structure Appearance Extend Configuration People Reports Help

Breadcrumb Place block

Breadcrumbs System Breadcrumb Configure

Content Place block **Place block**

Page title core Content Configure

Tabs core Content Configure

Help Help Content Configure

Primary admin actions core Content Configure

Main page content System Content Configure

Sidebar first Place block

The screenshot shows the 'Block layout' configuration page in the Drupal admin interface. It displays a list of blocks and their placement regions. The 'Content' section is currently selected. A red arrow points from the 'Place block' button in the 'Content' section to another 'Place block' button in the 'Sidebar first' section below it.

Place block

+ Add custom block

Filter by block name

BLOCK	CATEGORY	OPERATIONS
Page title	core	Place block
Primary admin actions	core	Place block
Tabs	core	Place block
Block Module Example	Custom	Place block
Search form	Forms	Place block
User login	Forms	Place block
Help	Help	Place block
Recent comments	Lists (Views)	Place block

Forms Sidebar first Configure



You may have to clear the Drupal cache again to get the new block to show up in the list. After clicking "Place block," a "Configure block" screen appears. You can safely just click "Save block."

Configure block

X

Block description: Block Module Example

Title *

Block Module Example

Machine name:

blockmoduleexample [Edit]

Display title

Visibility

Pages

Not restricted

Roles

Not restricted

Content type

Not restricted

Vocabulary

Pages

Specify pages by using their paths. Enter one path per line.
The '*' character is a wildcard. An example path is /user/*
for every user page. <front> is the front page.

Show for the listed pages

Hide for the listed pages

Save block

Navigate back to the home page of the site and you'll see your block appearing. Screenshot below:

In ddev93.ddev.site/user/... Apps Misc Clipboard Noah A Clients A Check out A Current Stuff Messages for web

Manage Shortcuts admin

Content Structure Appearance Extend Configuration People R

Drush Site-Install

Home

Home

Search

admin

View Shortcuts Edit

Tools

Add content

Block Module Example

It works!

Member for 2 hours 20 minutes

Your example block

The screenshot shows the Drupal 8 Admin interface. At the top, there's a navigation bar with links like 'Content', 'Structure', 'Appearance', 'Extend', 'Configuration', 'People', and 'R'. Below that is a main header with the 'Drush Site-Install' logo and the word 'admin'. On the left, there's a sidebar with 'Tools' and an 'Add content' button. The main content area shows a user profile for 'admin' with buttons for 'View', 'Shortcuts', and 'Edit'. Below the profile, there's a block titled 'Block Module Example' with the text 'It works!'. A red arrow points from the text 'Your example block' to this title. Further down, there's a message 'Member for 2 hours 20 minutes'. At the bottom of the page, there's a section for 'Primary admin actions' with three items: 'Block Module Example', 'Main page content', and 'Sidebar first Place block'. The 'Block Module Example' item has a dropdown menu with options: 'Configure', 'Disable', 'Remove', and 'Configure' again. A red arrow points from the text 'Remove the new block' to the 'Remove' option in this menu.

You can safely remove the block via the block layout page, choose "remove" from the dropdown next to your "Block Module Example"

Primary admin actions core Content Configure

Block Module Example Custom Content Configure Disable Remove Configure

Main page content System Content

Sidebar first Place block Remove the new block

This screenshot shows the 'Block Layout' page for the 'Block Module Example' block. It lists three blocks: 'Primary admin actions', 'Block Module Example', and 'Main page content'. The 'Block Module Example' block is highlighted. To its right is a configuration dropdown menu with four options: 'Configure', 'Disable', 'Remove', and 'Configure' again. A red arrow points from the text 'Remove the new block' to the 'Remove' option in this menu. At the bottom of the page, there are two buttons: 'Sidebar first' and 'Place block'.

Anatomy of a custom block with dependency injection

The block class PHP file is usually in <Drupal web root>/modules/custom/mymodule/src/Plugin/Block.

e.g. dev1/web/modules/custom/image_gallery/src/Plugin/Block/ImageGalleryBlock.php

or

dev1/web/modules/contrib/examples/block_example/src/Plugin/Block/ExampleConfigurableTextBlock.php

Specify namespace:

```
namespace Drupal\abc_wea\Plugin\Block;
```

Blocks always extend BlockBase but can also implement other interfaces... see below.

```
Class ImageGalleryBlock extends BlockBase
```

If you want to use Dependency Injection, implement: ContainerFactoryPluginInterface

e.g.

```
class ImageGalleryBlock extends BlockBase implements  
ContainerFactoryPluginInterface {
```

Be sure to include:

```
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
```

And for annotation translation:

```
use Drupal\Core\Annotation\Translation;
```

You can annotate like this:

```
/**  
 * Hello World Salutation block.  
 *  
 * @Block(  
 *   id = "hello_world_salutation_block",  
 *   admin_label = @Translation("Hello world salutation"),  
 *   category = @Translation("Custom")  
 * )  
 */
```

Or like this:

```
/**  
 * Provides an image gallery block.  
 *  
 * @Block(  
 *   id = "ig_product_image_gallery",  
 *   admin_label = @Translation("Product Image Gallery"),  
 *   category = @Translation("Image Display"),  
 *   context = {  
 *     "node" = @ContextDefinition(  
 *       "entity:node",  
 *       label = @Translation("Current Node")  
 *     )  
 *   }  
 * )  
 */
```

In most cases you will implement ContainerFactoryPluginInterface. Plugins require this for dependency injection. So don't forget:

```
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;  
  
class HelloWorldSalutationBlock extends BlockBase implements ContainerFactoryPluginInterface {
```

If you want dependency injection, you will need a create() function.

This will call the constructor (to do lazy loading) and call the container to ->get() the service you need. In the example below \$container->get('hello_world.salutation') does the trick. return new static() calls your class constructor.

Be sure to add your service to the list of parameters in the constructor: \$container->get('hello_world.salutation').

```
/**
 * {@inheritDoc}
 */
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    return new static(
        $configuration,
        $plugin_id,
        $plugin_definition,
        $container->get('hello_world.salutation')
    );
}
```

Here are your __construct() and a build() functions. See the 4th param – HelloWorldSalutationService \$salutation – that's the injected service.

```
/**
 * Construct.
 *
 * @param array $configuration
 *   A configuration array containing information about the plugin instance.
 * @param string $plugin_id
 *   The plugin_id for the plugin instance.
 * @param string $plugin_definition
 *   The plugin implementation definition.
 * @param \Drupal\hello_world>HelloWorldSalutation $salutation
 */
public function __construct(array $configuration, $plugin_id, $plugin_definition, HelloWorldSalutationService $salutation) {
    parent::__construct($configuration, $plugin_id, $plugin_definition);
    $this->salutation = $salutation;
}
```

```
/**
 * {@inheritDoc}
 */
public function build() {
    return [
        '#markup' => $this->salutation->getSalutation(),
    ];
}
```

TODO: NEED A BETTER EXAMPLE OF A D.I. BLOCK HERE especially showing a build()

Create a block with an entityQuery

You often need to query some data from Drupal and display it in a block.

Here is a simple block that loads all published content of type “page” and renders the titles. You could sort them by creation date by adding this to the \$query variable: ->sort('created', 'DESC');

```

namespace Drupal\opinions\Plugin\Block;

use Drupal\Core\Block\BlockBase;
use Drupal\Core\Annotation\Translation;

/**
 * Provides OpinionLanding Block.
 *
 * @Block(
 *   id = "opinion_landing",
 *   admin_label = @Translation("Opinion landing block"),
 * )
 *
 * @package Drupal\oag_opinions\Plugin\Block
 */
class OpinionLanding extends BlockBase {

  public function build() {

    $entity_type = 'node';
    $storage = \Drupal::entityTypeManager()->getStorage($entity_type);
    $query = \Drupal::entityQuery('node')
      ->condition('type', 'page')
      ->condition('status', 1);
    $nids = $query->execute();
    $nodes = $storage->loadMultiple($nids);

    $render_array = [];
    foreach ($nodes as $node) {
      $render_array[] = [
        '#type' => 'markup',
        '#markup' => '<p>' . $node->getTitle(),
      ];
    }

    return $render_array;
  }
}

```

Create a Block with a corresponding config form

Here is an example which includes a block and a corresponding config form that controls what is displayed in the block. The block can be placed using the Block Layout system in Drupal at /admin/structure/block (shown below) or via twig in a template file.

BLOCK	CATEGORY	REGION	OPERATIONS
Header	Place block	Header	Configure
Site branding	System		
Primary menu	Place block	Primary menu	Configure
Main navigation	Menus		

The config form definition

The config form is defined in `docroot/modules/custom/quick_pivot/src/Form/QuickPivotConfigForm.php` with a class which extends `ConfigFormBase` because this form is there for configuring its block:

```
class QuickPivotConfigForm extends ConfigFormBase {
```

In the class are the `getFormId()`, `getEditableConfigName()`, `buildForm()` and `submitForm()` functions which are all pretty straightforward.

The routing.yml file

Then in `docroot/modules/custom/quick_pivot/quick_pivot.routing.yml` we specify the route where we invoke the form.

Besides the `quick_pivot.info.yml` (module info) file, that should be all you need to make the config for the block.

The Block definition

Now for the block that users see (also the one that pops up in the block configuration) in `docroot/modules/custom/quick_pivot/src/Plugin/Block/QuickPivotSubscribeBlock.php`

We define the block with its annotation:

```
/*
 * Provides a cart block.
 *
 * @Block(
 *   id = "quick_pivot_subscribe_block",
 *   admin_label = @Translation("QuickPivot Subscribe Block"),
 *   category = @Translation("QuickPivot Subscribe")
 * )
 */
class QuickPivotSubscribeBlock extends BlockBase implements ContainerFactoryPluginInterface {
```

It implements `ContainerFactoryPluginInterface` to allow dependency injection. This is critical for plugins or blocks. More at <https://chromatichq.com/blog/dependency-injection-drupal-8-plugins>. All this interface defines is the `create()` method. Because we are using dependency injection, we need both a `create()` and a `__construct()`.

Here is the `create()`

```
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    return new static(
        $configuration,
        $plugin_id,
        $plugin_definition,
        $container->get('config.factory'),
        $container->get('form_builder')
    );
}
```

Here is the constructor:

```
public function __construct(array $configuration, $plugin_id, $plugin_definition, ConfigFactoryInterface $config_factory, FormBuilderInterface $form_builder) {
    parent::__construct($configuration, $plugin_id, $plugin_definition);

    $this->configFactory = $config_factory;
    $this->formBuilder = $form_builder;
}
```

And finally the `build()` method:

```
public function build() {
    return $this->formBuilder->getForm('Drupal\quick_pivot\Form\QuickPivotSubscribeForm');
}
```

Here is the docroot/modules/custom/quick_pivot/src/Form/QuickPivotSubscribeForm.php:

```
<?php

namespace Drupal\quick_pivot\Form;

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Form\FormBuilderInterface;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\ReplaceCommand;
use Drupal\Core\Ajax\CssCommand;
use Drupal\Core\Ajax\HtmlCommand;
use Drupal\Core\Ajax\AppendCommand;
use Drupal\quick_pivot\QuickPivotApiInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Provides a form for users to subscribe to QuickPivot.
 */
class QuickPivotSubscribeForm extends FormBase {

    /**
     * {@inheritDoc}
     */
    public function getFormId() {
        return 'quick_pivot_subscribe_form';
    }

    /**
     * {@inheritDoc}
     */
}
```

```

/*
public function buildForm(array $form, FormStateInterface $form_state) {
  $form['#id'] = 'quick-pivot-subscribe-form';
  $form['#cache'] = ['max-age' => 0];
  $form['#attributes'] = ['autocomplete' => 'off'];

  $form['email'] = [
    '#type' => 'textfield',
    '#id' => 'quick-pivot-email',
    '#placeholder' => $this->t('Email address'),
    '#attributes' => ['class' => ['edit-quick-pivot-email']],
    '#prefix' => '<div class="subscriber-email-msg">',
    '#suffix' => '</div>',
  ];
  $form['actions'][subscribe_submit] = [
    '#type' => 'submit',
    '#value' => $this->t('Sign Up'),
    '#name' => 'quick_pivot_subscribe_form_submit_button',
    '#ajax' => [
      'callback' => 'Drupal\quick_pivot\Form\QuickPivotSubscribeForm::quickPivotAjaxSubmit',
      'wrapper' => 'quick-pivot-subscribe-form',
      'progress' => ['type' => 'throbber', 'message' => NULL],
    ],
  ];
  $form['message'] = [
    '#type' => 'markup',
    '#markup' => '<div id="quick-pivot-message-area"></div>',
  ];

  return $form;
}

/**
 * {@inheritDoc}
 */
public function validateForm(array &$form, FormStateInterface $form_state) {

}

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state) {

}

/**
 * {@inheritDoc}
 */
public static function quickPivotAjaxSubmit(array &$form, FormStateInterface $form_state) {
  $validate = TRUE;
  $email = trim($form_state->getValue('email'));
  if (!$filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $message = t('Please enter a valid email address.');
    $validate = FALSE;
    $css_border = ['border' => '1px solid red'];
    $css_color = ['color' => 'red'];
  }
  if ($validate) {
    $css_border = ['border' => '1px solid green'];
  }
}

```

```

$css_border = [border => 1px solid green];
$css_color = ['color' => 'green'];
$response = \Drupal::service('quick_pivot.api')->subscribeEmail($email);
if (strpos(reset($response), 'Success') !== FALSE) {
  $message = t('Thank you for signing up. Your subscription has been activated.');
}
else {
  $message = t('Your subscription could not be processed.');
}

$response = new AjaxResponse();

$quick_pivot_form = \Drupal::formBuilder()->rebuildForm('quick_pivot_subscribe_form', $form_state);
if ($validate) {
  $quick_pivot_form['email'][ '#value' ] = '';
  $quick_pivot_form['email'][ '#placeholder' ] = t('Email address');
}
$response->addCommand(new ReplaceCommand('#quick-pivot-subscribe-form', $quick_pivot_form));
$response->addCommand(new CssCommand('#edit-quick-pivot-email', $css_border));
$response->addCommand(new HtmlCommand('#quick-pivot-message-area', $message));
$response->addCommand(new CssCommand('#quick-pivot-message-area', $css_color));
return $response;
}
}

```

Here is the entire QuickPivotConfigForm.php file:

```

<?php

namespace Drupal\quick_pivot\Form;

use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Form\ConfigFormBase;

/**
 * Configure Websphere settings for this site.
 */
class QuickPivotConfigForm extends ConfigFormBase {

  /**
   * {@inheritDoc}
   */
  public function getFormId() {
    return 'quick_pivot_settings';
  }

  /**
   * {@inheritDoc}
   */
  protected function getEditableConfigNames() {
    return ['quick_pivot.settings'];
  }

  /**
   * {@inheritDoc}
   */
  public function buildForm(array $form, FormStateInterface $form_state) {

```

```

$config = $this->config('quick_pivot.settings');

$form['quick_pivot_settings'] = [
  '#type' => 'details',
  '#title' => $this->t('Quick Pivot API Settings'),
  '#open' => TRUE,
  '#weight' => 1,
];

$form['quick_pivot_settings']['api_end_point'] = [
  '#type' => 'textfield',
  '#title' => $this->t('API End point'),
  '#description' => $this->t("Enter the API end point URL."),
  '#default_value' => $config->get('quick_pivot_settings.api_end_point'),
  '#required' => TRUE,
  '#size' => 100,
];

$form['quick_pivot_settings']['user_guid'] = [
  '#type' => 'textfield',
  '#title' => $this->t('User GUID'),
  '#description' => $this->t("SOAP API User GUID"),
  '#default_value' => $config->get('quick_pivot_settings.user_guid'),
  '#required' => TRUE,
  '#size' => 100,
];

$form['quick_pivot_settings']['account'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Account'),
  '#description' => $this->t("SOAP API Account"),
  '#default_value' => $config->get('quick_pivot_settings.account'),
  '#required' => TRUE,
  '#size' => 100,
];

$form['quick_pivot_settings']['sender'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Sender'),
  '#description' => $this->t("SOAP API Sender"),
  '#default_value' => $config->get('quick_pivot_settings.sender'),
  '#required' => TRUE,
  '#size' => 100,
];

return parent::buildForm($form, $form_state);
}

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state) {

  $this->config('quick_pivot.settings')
    ->set('quick_pivot_settings.api_end_point', $form_state->getValue('api_end_point'))
    ->set('quick_pivot_settings.user_guid', $form_state->getValue('user_guid'))
    ->set('quick_pivot_settings.account', $form_state->getValue('account'))
    ->set('quick_pivot_settings.sender', $form_state->getValue('sender'))
    ->save();
}

```

```

parent::submitForm($form, $form_state);
}

}

```

And the QuickPivotSubscribeBlock.php:

```

<?php

namespace Drupal\quick_pivot\Plugin\Block;

use Drupal\Core\Block\BlockBase;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\Core\Config\ConfigFactoryInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Form\FormBuilderInterface;

/**
 * Provides a cart block.
 *
 * @Block(
 *   id = "quick_pivot_subscribe_block",
 *   admin_label = @Translation("QuickPivot Subscribe Block"),
 *   category = @Translation("QuickPivot Subscribe")
 * )
 */
class QuickPivotSubscribeBlock extends BlockBase implements ContainerFactoryPluginInterface {

 /**
 * The configuration factory.
 *
 * @var \Drupal\Core\Config\ConfigFactoryInterface
 */
 protected $configFactory;

 /**
 * The form builder.
 *
 * @var \Drupal\Core\Form\FormBuilderInterface
 */
 protected $formBuilder;

 /**
 * Constructor for the QuickPivot subscribe block.
 *
 * @param array $configuration
 *   The block configuration.
 * @param string $plugin_id
 *   The plugin_id for the plugin instance.
 * @param mixed $plugin_definition
 *   The plugin implementation definition.
 * @param \Drupal\Core\Config\ConfigFactoryInterface $config_factory
 *   The configuration factory.
 * @param \Drupal\Core\Form\FormBuilderInterface $form_builder
 *   The form builder.
 */
 public function __construct(array $configuration, $plugin_id, $plugin_definition, ConfigFactoryInterface $config_factory, FormBuilderInterface $form_builder) {
 parent::__construct($configuration, $plugin_id, $plugin_definition);
}

```

```

    $this->configFactory = $config_factory;
    $this->formBuilder = $form_builder;
}

/**
 * {@inheritDoc}
 */
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    return new static(
        $configuration,
        $plugin_id,
        $plugin_definition,
        $container->get('config.factory'),
        $container->get('form_builder')
    );
}

/**
 * Builds the cart block.
 *
 * @return array
 *   A render array.
 */
public function build() {
    return $this->formBuilder->getForm('Drupal\quick_pivot\Form\QuickPivotSubscribeForm');
}

}

```

And here is the routing file: docroot/modules/custom/quick_pivot/quick_pivot.routing.yml

```

quick_pivot.config:
path: '/admin/config/quick_pivot/settings'
defaults:
_form: 'Drupal\quick_pivot\Form\QuickPivotConfigForm'
_title: 'Quick Pivot Settings'
requirements:
_permission: 'administer site configuration'

```

And for the icing, We also specify a menu item so users can access the configuration form via the menu system at docroot/modules/custom/quick_pivot/quick_pivot.links.menu.yml.

```

quick_pivot.config:
title: 'QuickPivot API settings'
description: 'Configure the QuickPivot API Settings.'
parent: system.admin_config_services
route_name: quick_pivot.config
weight: 1

```

Modify a block with hook_block_view_alter or hook_block_build_alter

Some drupal hooks only run inside a contributed modules and some only inside a theme and some both.

```

function themename_preprocess_block(&$variables)
{
  if ($variables['plugin_id'] == 'entity_browser_block:department_info') {
    $variables['#attached']['library'][] = 'drupal/libraryname';
  }
}

```

What's described below could potentially be done on a theme preprocess for the block.

If you need to modify a block, you can supposedly use `hook_block_view_alter` or `hook_block_build_alter`, although I haven't been able to make this work... hmm.

There is a comment that may be worth exploring at

https://api.drupal.org/api/drupal/core%21modules%21block%21block.api.php/function/hook_block_view_alter/8.2.x.

To alter the block content you must add a `#pre_render` in the `hook_block_view_alter` hook.

In <https://drupal.stackexchange.com/a/215948> there is an example which fills in the `$build['#pre_render']` array with a string.

In an example on that stackexchange site, this function is provided:

```
function yourmodule_block_view_alter(array &$build, \Drupal\Core\Block\BlockPluginInterface $block) {
  if ($block->getBaseId() === 'system_powered_by_block') {
    $build['#pre_render'][] = '_yourmodule_block_poweredby_prerender';
  }
}
```

I think this is the version I tried:

```
/** 
 * Implements hook_block_build_alter().
 */
function plug_academy_core_block_build_alter(array &$build, \Drupal\Core\Block\BlockPluginInterface $block) {
  if ($block->getPluginId() == 'system_menu_block:account') {
    $build['#cache']['contexts'][] = 'url';
  }
  // else if ($block->getBaseId() === 'block_content') {
  //   if ($block->label() == "Home Page Alert") {
  //     $build['content'] = '<p>New content built here!</p>';
  //   }
  // }
  // }
}
```

And I discovered an example from a project where the `$build['#pre_render']` array is populated with a function. I'm not sure what that function did – presumably returned some text to be rendered.

```
/** 
 * Implements hook_block_view_alter().
 */
function pega_academy_core_block_view_alter(array &$build, \Drupal\Core\Block\BlockPluginInterface $block) {
  if ($block->getBaseId() === 'block_content') {
    if ($block->label() == "Home Page Alert") {
      $build['#pre_render'][] = 'Drupal\pega_academy_core\Controller\DashboardController::home_page_alert_prerender';
    }
    $build['content'] = '<p>New content built here!</p>';
  }
}
```

Disable caching in a block

From `docroot/modules/custom/websphere_commerce/modules/cart/src/Plugin/Block/CartSummary.php`:

```
/**
 * {@inheritDoc}
 */
public function getCacheMaxAge() {
  return 0;
}
```

Add a configuration form to your block

Making a block configurable means it has a form where you can specify its settings, e.g., the configuration form for the menu block module allows you to specify menu levels. Ignore this if your block does not need any configuration.

To make your block configurable, override 3 methods from BlockBase.

- 1 defaultConfiguration
- 2 blockForm
- 3 blockSubmit

Here `defaultConfiguration()` returns a `block_count` of 5.

```
/**
 * {@inheritDoc}
 */
public function defaultConfiguration() {
  // By default, the block will display 5 thumbnails.
  return [
    'block_count' => 5,
  ];
}
```

`blockForm()` is used to create a configuration form:

```
/**
 * {@inheritDoc}
 */
public function blockForm($form, FormStateInterface $form_state) {
  $range = range(2, 20);
  $form['block_count'] = [
    '#type' => 'select',
    '#title' => $this->t('Number of product images in block'),
    '#default_value' => $this->configuration['block_count'],
    '#options' => array_combine($range, $range),
  ];
  return $form;
}
```

And `blockSubmit()` handles the submission of the config form. You don't need to save anything. The data is saved automatically into the Drupal config system. You just specify a configuration key like `$this->configuration['block_count']` and the rest is handled for you.

```
/**
 * {@inheritDoc}
 */
public function blockSubmit($form, FormStateInterface $form_state) {
  $this->configuration['block_count'] = $form_state->getValue('block_count');
}
```

The `build()` method does all the work of building a render array to display whatever your block wants to display. Here is an example of a

build() function.

```
/*
 * {@inheritDoc}
 */
public function build() {
  $build = [];
  $node = $this->getContextValue('node');

  // Determine if we are on a page that points to a product.
  $product = $this->getProduct($node);
  if ($product) {

    // Retrieve the product images
    $image_data = $this->productManagerService->retrieveProductImages($product);
    $block_count = $this->configuration['block_count'];
    $item_count = 0;
    $build['list'] = [
      '#theme' => 'item_list',
      '#items' => [],
    ];
    $build['list']['#items'][0] = [
      '#type' => 'markup',
      '#markup' => $this->t('There were no product images to display.')
    ];

    while ($item_count < $block_count && isset($image_data[$item_count])) {
      $file = File::load($image_data[$item_count]['target_id']);
      $link_text = [
        '#theme' => 'image_style',
        '#uri' => $file->getFileUri(),
        '#style_name' => 'product_thumbnail',
        '#alt' => $image_data[$item_count]['alt'],
      ];
      // Modal dialog
      // see https://www.drupal.org/node/2488192 for more on modals
      $options = [
        'attributes' => [
          'class' => [
            'use-ajax',
          ],
          'data-dialog-type' => 'modal',
          'data-dialog-options' => Json::encode([
            'width' => 700,
          ]),
        ],
      ];
      $url = Url::fromRoute('abc_prg.display_product_image', ['node' => $product->nid->value, 'delta' => $item_count]);
      $url->setOptions($options);
      $build['list']['#items'][$item_count] = [
        '#type' => 'markup',
        '#markup' => Link::fromTextAndUrl($link_text, $url)
          ->toString(),
      ];
      $item_count++;
    }
    $build['#attached']['library'][] = 'core/drupal.dialog.ajax';
  }
}
```

```
    }
}

return $build;
}
```

One last item. Configuration expects a schema for things being saved. Here we create a .schema.yml in /config/schema and it looks like:

```
# Schema for the configuration files for my module.
```

```
block.settings.alchemy_block:
  type: block_settings
  label: 'Alchemy block'
  mapping:
    block_count:
      type: integer
      label: 'Block count'
```

Block display not updating after changing block content

From [Nedcamp video on caching by Kelly Lucas, November 2018](#)

In a twig template, if you just want to render one or more fields (instead of the entire node), Drupal may not be aware if the content has changed, and will sometimes show old cached content. To resolve this, define a view mode and call `content | render` and assign the result to a variable like this:

```
set blah = content|render
```

Be sure to surround the above code with curly brace and percentage sign delimiters. Unfortunately these don't always render correctly in this document so I've had to remove them for now.

Adding this render call will cause Drupal to render the content for that node, which will cause a check of the caches and make sure the most current content is rendered.

Then add your fields:

```
{content.field_one} etc.
```

Block Permission (blockAccess)

This code is taken from the Drupal core user_login_block (UserLoginBlock.php). It allows access to the block if the user is logged out and is not on the login or logout page. The access is cached based on the current route name and the user's current role being anonymous. If these are not passed, the access returned is forbidden and the block is not built.

```

use Drupal\Core\Access\AccessResult;

$account = \Drupal::currentUser();

/**
 * {@inheritDoc}
 */
protected function blockAccess(AccountInterface $account) {
  $route_name = $this->routeMatch->getRouteName();
  if ($account->isAnonymous() && !in_array($route_name, ['user.login', 'user.logout'])) {
    return AccessResult::allowed()
      ->addCacheContexts(['route.name', 'user.roles:anonymous']);
  }
  return AccessResult::forbidden();
}

```

Another example from the Drupal core Copyright.php file:

```

// $account comes from
$account = \Drupal::currentUser();

//Get the route.
$route_name = \Drupal::routeMatch()->getRouteName();

// not on the user login and logout pages
if (!in_array($route_name, ['user.login', 'user.logout'])) {
  return AccessResult::allowed();
}

//Authenticated user
if ($account->isAuthenticated()) {
  return AccessResult::allowed();
}

//Anonymous user.
if ($account->isAnonymous()) {
  return AccessResult::forbidden();
}

```

Blocks shouldn't talk to the router, NodeRouteContext and friends should

While it is possible for blocks to talk to the router, you can't always count that they will be on a meaningful route i.e. are they being displayed on a node? So we should use context definition in the block annotation like this:

```

/**
 * Provides a 'Node Context Test' block.
 *
 * @Block(
 *   id = "node_block_test_context",
 *   label = @Translation("Node Context Test"),
 *   context_definitions = {
 *     "node" = @ContextDefinition("entity:node", label = @Translation("Node"))
 *   }
 * )
 */

```

This causes the block to be available only on various node pages (view, edit etc.). This can be changed:

```

* context_definitions = {
*   "node" = @ContextDefinition("entity:node", label = @Translation("Node"),
*     required = FALSE)
* }

```

The order of named options passed to ContextDefinition after the first argument does not matter.

Then in the block we check to make sure the user is viewing a node and that the user has view rsvlist permission. See the code below:

```

protected function blockAccess(AccountInterface $account) {
  /** @var \Drupal\Core\Plugin\Context\Context $node */
  $cacheContext = $this->getContext('node');
  /** @var \Drupal\Core\Entity\Plugin\DataType\EntityAdapter $data */
  $data = $cacheContext->getContextData();
  /** @var \Drupal\node\NodeInterface $node */
  $node = $data->getValue();
  if ($node) {
    $nid = $node->id();
    if (is_numeric($nid)) {
      // See rsvp.permissions.yml for the permission string.
      return AccessResult::allowedIfHasPermission($account, 'view rsvlist');
    }
  }
  return AccessResult::forbidden();
}

```

More at <https://drupal.stackexchange.com/questions/145823/how-do-i-get-the-current-node-id/314152#314152>

Note. While this practice is not recommended, the RSVP module does have an example of a block talking to the router i.e. \Drupal::routeMatch() - see https://git.drupalcode.org/project/rsvp_module/-/blob/1.0.x/src/Plugin/Block/RSVPBlock.php where the blockAccess() function grabs the node parameter and acts on it.

```

/**
 * {@inheritDoc}
 */
public function blockAccess(AccountInterface $account) {
  /** @var \Drupal\node\Entity\Node $node */
  $node = \Drupal::routeMatch()->getParameter('node');
  $nid = $node->nid->value;
  /** @var \Drupal\rsvp_module\EnablerService $enabler */
  $enabler = \Drupal::service('rsvp_module.enabler');
  if(is_numeric($nid)) {
    if($enabler->isEnabled($node)) {
      return AccessResult::allowedIfHasPermission($account, 'view rsvp_module');
    }
  }
  return AccessResult::forbidden();
}

```

Values returned by blockAccess()

Some options that can be returned from blockAccess() are:

```

return AccessResult::forbidden();
return AccessResult::allowed();
return AccessResult::allowedIf(TRUE);

```

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) 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.

Caching and cache tags

TABLE OF CONTENTS

- [How to uncache a particular page or node](#)
- [Don't cache data returned from a controller](#)
- [Disable caching for a content type](#)
- [Considering caching when retrieving query, get or post parameters](#)
- [Debugging Cache tags](#)
- [Using cache tags](#)
- [Setting cache keys in a block](#)
- [Getting Cache Tags and Contexts for a block](#)
- [Caching REST Resources](#)
- [Caching in an API class wrapper](#)
- [Caching in a .module file](#)
- [Logic for caching render arrays](#)
- [Development Setup](#)
 - [Disable caching and enable TWIG debugging](#)
 - [Disable Cache for development](#)
- [How to specify the cache backend](#)
 - [class ChainedFastBackend](#)
 - [APCu](#)
- [Reference](#)

views | 274

How to uncache a particular page or node

This will cause Drupal to rebuild the page internally, but won't stop browsers or CDN's from caching.

```
\Drupal::service('page_cache_kill_switch')->trigger();
```

Use this statement in node_preprocess, controller, etc.

Create a custom module to implement setting max-age to 0. For example in ddd.module file:

```
use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Entity\Display\EntityViewDisplayInterface;

function ddd_node_view_alter(array &$build, EntityInterface $entity, EntityViewDisplayInterface $display) {
  $bundle = $entity->bundle();
  if ($bundle == 'search_home') {
    $build['#cache']['max-age'] = 0;
    \Drupal::service('page_cache_kill_switch')->trigger();
  }
}
```

Don't cache data returned from a controller

From dev1/web/modules/custom/rsvp/src/Controller/ReportController.php

```
// Don't cache this page.  
$content['#cache']['max-age'] = 0;  
return $content;  
}
```

Disable caching for a route in the module.routing.yml file.

```
requirements:  
  _permission: 'access content'  
options:  
  no_cache: TRUE
```

Disable caching for a content type

If someone tries to view a node of content type `search_home` (i.e. an entity of bundle `search_home`) caching is disabled and Drupal and the browser will always re-render the page. This is necessary for a page that is retrieving data from a third party source and you almost always expect it to be different. It wouldn't work for a search page to show results from a previous search.

```
use Drupal\Core\Entity\EntityInterface;  
use Drupal\Core\Entity\Display\EntityViewDisplayInterface;  
  
/**  
 * Implements hook_ENTITY_TYPE_view_alter().  
 */  
function ddd_node_view_alter(array &$build, EntityInterface $entity, EntityViewDisplayInterface $display) {  
  $bundle = $entity->bundle();  
  if ($bundle == 'search_home') {  
    $build['#cache']['max-age'] = 0;  
    \Drupal::service('page_cache_kill_switch')->trigger();  
  }  
}
```

Considering caching when retrieving query, get or post parameters

For get variables use:

```
$query = \Drupal::request()->query->get('name');
```

For post variables use:

```
$name = \Drupal::request()->request->get('name');
```

For all items in get:

```
$query = \Drupal::request()->query->all();  
$search_term = $query['query'];  
$collection = $query['collection'];
```

Be wary about caching. From <https://drupal.stackexchange.com/questions/231953/get-in-drupal-8/231954#231954> the code provided only works the first time so it is important to add a '#cache' context in the markup.

```

namespace Drupal\newday\Controller;

use Drupal\Core\Controller\ControllerBase;

class NewdayController extends ControllerBase {
  public function new() {
    $day = [
      "#markup" => \Drupal::request()->query->get('id'),
    ];
    return $day;
  }
}

```

The request is being cached, you need to tell the system to vary by the query argument:

```

$day = [
  '#markup' => \Drupal::request()->query->get('id'),
  '#cache' => [
    'contexts' => ['url.query_args:id'],
  ],
];

```

More about caching render arrays at <https://www.drupal.org/docs/8/api/render-api/cacheability-of-render-arrays>

Debugging Cache tags

In development.services.yml set these parameters

```

parameters:
  http.response.debug_cacheability_headers: true

```

in Chrome, the network tab, click on the doc and view the Headers. You will see the following two headers showing both the cache contexts and the cache tags

1 X-Drupal-Cache-Contexts:

languages:language_interface route session theme timezone url.path url.query_args url.site user

1 X-Drupal-Cache-Tags:

block_view config:block.block.bartik_account_menu config:block.block.bartik_branding config:block.block.bartik_breadcrumbs config:block.block.bartik_content config:block.block.bartik_footer config:block.block.bartik_help config:block.block.bartik_local_actions config:block.block.bartik_local_tasks config:block.block.bartik_main_menu config:block.block.bartik_messages config:block.block.bartik_page_title config:block.block.bartik_powered config:block.block.bartik_search config:block.block.bartik_tools config:block.block.helloworldsalutation config:block.block.modalblock config:block.block.productimagegallery config:block.block.rsvpblock config:block.block.views_block_aquifer_listing_block_1 config:block.block.views_block_related_videos_block_1 config:block.block.views_block_user_guide_pages_referencing_a_product_block_1 config:block.block.views_block_workshop_count_proposed_workshop_block config:bloc

Using cache tags

If you are generating a list of cached node teasers and you want to make sure your list is always accurate, use cache tags. To refresh the list every time a node is added, deleted or edited you could use a render array like this:

```
$build = [
  '#type' => 'markup',
  '#markup' => $sMarkup,
  '#cache' => [
    'keys' => ['home-all','home'],
    'tags'=> ['node_list'], // invalidate cache when any node content is added/changed etc.
    'max-age' => '36600', // invalidate cache after 10h
  ],
];
```

It is possible to change this so the cache is invalidated only when a content type of *book* or *magazine* is changed in two possible ways:

- 1 Include all node tags (node:{#id}), if doesn't matter if a new node of a particular type was added.
- 2 Create and control your own cache tag, and invalidate it when you want.

If you want a block to be rebuilt every time that a term from a particular vocab_id is added, changed, or deleted you can cache the term list. If you need to cache a term list per vocab_id - i.e. every time that a term from a particular vocab_id is added, changed, or deleted the cache tag is invalidated using Cache::invalidateTags(\$tag_id) then my block will be rebuilt.

```
use Drupal\Core\Cache\Cache;

function filters_invalidate_vocabulary_cache_tag($vocab_id) {
  Cache::invalidateTags(['filters_vocabulary' . $vocab_id]);
}
```

If you want this to work for nodes, you may be able to just change \$vocab_id for \$node_type.

Setting cache keys in a block

If you add some code to a block that includes the logged in user's name, you may find that the username will not be displayed correctly – rather it may show the prior users name. This is because the cache context of user doesn't bubble up to the display of the container (e.g. the node that is displayed along with your custom block.) Add this to bubble the cache contexts up.

```
public function getCacheContexts() {
  Return Cache::mergeContexts(parent::getCacheContexts(),['user']);
}
```

and scrolling down a bit at this link shows some more info about getting cache tags and merging them.

<https://drupal.stackexchange.com/questions/145823/how-do-i-get-the-current-node-id>

Getting Cache Tags and Contexts for a block

In this file /modules/custom/dart_pagination/src/Plugin/Block/VideoPaginationBlock.php I have a block that renders a form. The form queries some data from the database and will need to be updated depending on the node that I am on.

I added the following two functions:

```

public function getCacheTags() {
    //When my node changes my block will rebuild
    if ($node = \Drupal::routeMatch()->getParameter('node')) {
        //if there is node add its cachetag
        return Cache::mergeTags(parent::getCacheTags(), ['node:' . $node->id()]);
    } else {
        //Return default tags instead.
        return parent::getCacheTags();
    }
}

public function getCacheContexts() {
    //if you depend on \Drupal::routeMatch()
    //you must set context of this block with 'route' context tag.
    //Every new route this block will rebuild
    return Cache::mergeContexts(parent::getCacheContexts(), ['route']);
}

```

Caching REST Resources

Interesting article about caching REST resources at <http://blog.dcycle.com/blog/2018-01-24/caching-drupal-8-rest-resource/>

We can get Drupal to cache our rest resource e.g. in dev1 /custom/iai_wea/src/Plugin/rest/resource/WEAResource.php where we add this to our response:

```

if (!empty($record)) {
    $response = new ResourceResponse($record, 200);
    $response->addCacheableDependency($record);
    return $response;
}

```

Caching in an API class wrapper

From docroot/modules/custom/cm_api/src/CmAPIClient.php

Here a member is set up in the class

```

/**
 * Custom service to call APIs.
 *
 * @see \Drupal\cm_api\CmAPIClientInterface
 */
class CmAPIClient implements CmAPIClientInterface {

    ...
    /**
     * Internal static cache.
     *
     * @var array
     */
    protected static $cache = [];
}

```

The api call is made and the cache is checked. The index (or key) is built from the api call “getPolicy” and the next key is the policy number with the version number attached. So the \$response_data is put in the cache with:

```
self::$cache['getPolicy'][$policy_number . $version] = $response_data;
```

and retrieved with:

```
$response_data = self::$cache['getPolicy'][$policy_number . $version];
```

This relieves the back end load by rather getting the data from the cache if it is in the cache. (If the cache is warm.)

The entire function is shown below. It is from docroot/modules/custom/cm_api/src/CmAPIClient.php:

```
public function getPolicy($policy_number, $version = 'v2') {
    // Api action type.
    $this->params['api_action'] = 'Get policy';
    // Add policy number to display in watchdog.
    $this->params['policynumber'] = $policy_number;
    $base_api_url = $this->getBaseApiUrl();
    if (empty($policy_number) || !is_numeric($policy_number)) {
        $this->logger->get('cm_api_get_policy')
            ->error('Policy number must be a number.');
        return FALSE;
    }
    $endpoint_url = $base_api_url . '/' . $version . '/Policies/group?policies=' . $policy_number . '&include_ref=false&include_hist=true';

    if (!isset(self::$cache['getPolicy'][$policy_number . $version])) {
        $response_data = self::$cache['getPolicy'][$policy_number . $version];
    } else {
        $response_data = $this->performRequest($endpoint_url, 'GET', $this->params);
        self::$cache['getPolicy'][$policy_number . $version] = $response_data;
    }
    return $response_data;
}
```

Caching in a .module file

From docroot/modules/custom/ncs_infoconnect/nzz_zzzzconnect.module.

In the hook_preprocess_node function, we are calling an api to get some data.

```
/**
 * Implements hook_preprocess_node().
 */
function nzz_zzzzconnect_preprocess_node(&$variables) {
```

Notice the references to \Drupal::cache(). First we check if this is our kind of node to process. Then we derive the \$cid. We check the cache with a call to ->get(\$cid) and if it fails we:

- 1 call the api with \$client->request('GET')
- 2 pull out the body with \$response->getBody()
- 3 set the whole body into the cache with:

```
\Drupal::cache()->set($cid, $contents, REQUEST_TIME + (300));
```

In future requests, we can just use the data from the cache.

```

if ($node_type == 'zzzzfeed' && $published) {

$uuid = $variables[node]->field_uuid->getValue();
$nid = $variables[node]->id();

$nzz_auth_settings = Settings::get('nzz_api_auth', []);
$uri = $ncs_auth_settings['default'][server] . ':' . $ncs_auth_settings['default'][port];
$uri .= '/blahcontent/search';
$client = \Drupal::httpClient();

$cid = 'zzzzfeed-' . $nid;

try {
  if ($cache = \Drupal::cache()->get($cid)) {
    $contents = $cache->data;
  }
  else {
    $response = $client->request('GET', $uri, [
      'auth' => [$nzz_auth_settings['default']['username'], $nzz_auth_settings['default']['password']],
      'query' => [
        'uuid' => $uuid[0][value],
      ],
      'timeout' => 1,
    ]);
    $contents = $response->getBody()->getContents();
    \Drupal::cache()->set($cid, $contents, REQUEST_TIME + (300));
  }
}
catch (RequestException $e) {
  watchdog_exception('nzz_zzzconnect', $e);
  return FALSE;
}
catch (ClientException $e) {
  watchdog_exception('nzz_zzzconnect', $e);
  return FALSE;
}

$contents = json_decode($contents, TRUE);
$body = $contents[hits][hits][0][versions][0][properties][Text][0];
$variables['content']['body'] = [
  '#markup' => $body,
];
}

```

Logic for caching render arrays

From <https://www.drupal.org/docs/8/api/render-api/cacheability-of-render-arrays>

Whenever you are generating a render array, use the following 5 steps:

- 1 I'm rendering something. That means I must think of cacheability.
- 2 Is this something that's expensive to render, and therefore is worth caching? If the answer is yes, then what identifies this particular representation of the thing I'm rendering? Those are the cache keys.
- 3 Does the representation of the thing I'm rendering vary per combination of permissions, per URL, per interface language, per ... something? Those are the cache contexts. Note: cache contexts are completely analogous to HTTP's Vary header.
- 4 What causes the representation of the thing I'm rendering become outdated? I.e., which things does it depend upon, so that when those things change, so should my representation? Those are the cache tags.
- 5 When does the representation of the thing I'm rendering become outdated? I.e., is the data valid for a limited period of time only? That is

the max-age (maximum age). It defaults to “permanently (forever) cacheable” (Cache::PERMANENT). When the representation is only valid for a limited time, set a max-age, expressed in seconds. Zero means that it’s not cacheable at all.

Cache contexts, tags and max-age must always be set, because they affect the cacheability of the entire response. Therefore they “bubble” and parents automatically receive them.

Cache keys must only be set if the render array should be cached.

There are more details at the link above

Development Setup

Disable caching and enable TWIG debugging

Generally I enable twig debugging and disable caching while developing a site.

To enable TWIG debugging output in source, in sites/default/development.services.yml set twig.config debug:true. See core.services.yml for lots of other items to change for development

```
# Local development services.  
#  
# To activate this feature, follow the instructions at the top of the  
# 'example.settings.local.php' file, which sits next to this file.  
  
parameters:  
  http.response.debug_cacheability_headers: true  
  dino.roar.use_key_value_cache: true  
  
twig.config:  
  debug: true  
  auto_reload: true  
  cache: false  
  
# To disable caching, you need this and a few other items  
services:  
  cache.backend.null:  
    class: Drupal\Core\Cache\NullBackendFactory
```

to enable put this in settings.local.php:

```
/**  
 * Enable local development services.  
 */  
$settings['container_yamls'][0] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in settings.local.php with:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';
```

Disable Cache for development

From <https://www.drupal.org/node/2598914>

- 1 Copy, rename, and move the sites/example.settings.local.php to sites/default/settings.local.php with:

```
$ cp sites/example.settings.local.php sites/default/settings.local.php
```

- 1 Edit sites/default/settings.php and uncomment these lines:

```
if (file_exists($app_root . '/' . $site_path . '/settings.local.php')) {  
  include $app_root . '/' . $site_path . '/settings.local.php';  
}
```

This will include the local settings file as part of Drupal's settings file.

- 1 In `settings.local.php` make sure `development.services.yml` is enabled with:

```
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

By default `development.services.yml` contains the settings to disable Drupal caching:

```
services:  
  cache.backend.null:  
    class: Drupal\Core\Cache\NullBackendFactory
```

NOTE: Do not create `development.services.yml`, it already exists under `/sites`. You can copy it from there.

- 1 In `settings.local.php` change the following to be TRUE if you want to work with enabled css- and js-aggregation:

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

- 1 Uncomment these lines in `settings.local.php` to disable the render cache and disable dynamic page cache:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';  
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';  
$settings['cache']['bins']['page'] = 'cache.backend.null';
```

If you do not want to install test modules and themes, set the following to FALSE:

```
$settings['extension_discovery_scan_tests'] = FALSE;
```

- 1 In `sites/development.services.yml` add the following block to disable the twig cache:

```
parameters:  
  twig.config:  
    debug: true  
    auto_reload: true  
    cache: false
```

NOTE: If the parameters block is already present in the yml file, append the twig.config block to it.

Afterwards rebuild the Drupal cache with `drush cr` otherwise your website will encounter an unexpected error on page reload.

How to specify the cache backend

This information is relevant for using [Memcache](#), [Redis](#) and also [APCu](#). By default, Drupal caches information in the database. Tables includes `cache_default`, `cache_render`, `cache_page`, `cache_config` etc. By using the configuration below, Drupal can instead store this info in memory to increase performance.

Summary

Drupal will no longer automatically use the custom global cache backend specified in `$settings['cache']['default']` in `settings.php` on certain specific cache bins that define their own `default_backend` in their service definition. In order to override the default backend, a line must be added explicitly to `settings.php` for each specific bin that provides a `default_backend`. This change has no effect for users that do not use a custom cache backend configuration like Redis or Memcache, and makes it possible to remove workarounds that were previously necessary to keep using the default fast chained backend for some cache bins defined in Drupal core.

Detailed description with examples

In Drupal 8 there are several ways to specify which cache backend is used for a certain cache bin (e.g. the discovery cache bin or the render cache bin).

In Drupal, cache bins are defined as services and are tagged with name: `cache.bin`. Additionally, some cache bins specify a `default_backend` service within the tags. For example, the discovery cache bin from Drupal core defines a fast chained default backend:

```
cache.discovery:  
  class: Drupal\Core\Cache\CacheBackendInterface  
  tags:  
    - { name: cache.bin, default_backend: cache.backend.chainedfast }  
  factory: cache_factory:get  
  arguments: [discovery]
```

Independent of this, the \$settings array can be used in settings.php to assign cache backends to cache bins. For example:

```
$settings['cache']['bins']['discovery'] = 'cache.backend.memory';  
$settings['cache']['default'] = 'cache.backend.redis';
```

Before Drupal 8.2.0, the order of steps through which the backend was selected for a given cache bin was as follows:

- First look for a specific bin definition in settings. E.g., \$settings['cache']['bins']['discovery']
- If not found, then use the global default defined in settings. I.e., \$settings['cache']['default']
- If a global default is not defined in settings, then use the default_backend from the tag in the service definition.

This was changed to:

- First look for a specific bin definition in settings. E.g., \$settings['cache']['bins']['discovery']
- If not found, then use the the default_backend from the tag in the service definition.
- If no default_backend for the specific bin was provided, then use the global default defined in settings. I.e., \$settings['cache']['default'] The old order resulted in unexpected behaviors, for example, the fast chained backend was no longer used when an alternative cache backend was set as default.

The order has been changed, so that the cache bin services that explicitly set default_backends are always used unless explicitly overridden with a per-bin configuration. In core, this means, fast chained backend will be used for bootstrap, config, and discovery cache bins and memory backend will be used for the static cache bin, unless they are explicitly overridden in settings.

For example, to ensure Redis is used for all cache bins, before 8.2.0, the following configuration would have been enough:

```
$settings['cache']['default'] = 'cache.backend.redis';
```

However, now the following configuration in settings.php would be required to achieve the same exact behavior:

```
$settings['cache']['bins']['bootstrap'] = 'cache.backend.redis';  
$settings['cache']['bins']['discovery'] = 'cache.backend.redis';  
$settings['cache']['bins']['config'] = 'cache.backend.redis';  
$settings['cache']['bins']['static'] = 'cache.backend.redis';  
$settings['cache']['default'] = 'cache.backend.redis';
```

Before proceeding to override the cache bins that define fast cached default backends blindly, please also read why they exist, particularly when using multiple webserver nodes. See [ChainedFastBackend on api.drupal.org](#).

The above information is from <https://www.drupal.org/node/2754947>

Fabian Franz in his article at <https://drupalsun.com/fabianx/2015/12/01/day-1-tweak-drupal-8-performance-use-apcu-24-days-performance-goodies> suggests that we can configure APCu to be used for caches with the following:

```
$settings['cache']['default'] = 'cache.backend.apcu';  
$settings['cache']['bins']['bootstrap'] = 'cache.backend.apcu';  
$settings['cache']['bins']['config'] = 'cache.backend.apcu';  
$settings['cache']['bins']['discovery'] = 'cache.backend.apcu';
```

WARNING

Proceed with caution with the above as it seems that APCu may only suitable for single server setups. TODO: I couldn't find any references to using APCu with multi-server setups so I'm not sure if that is a safe configuration.

Pantheon docs ask in their FAQ Can APCu be used as a cache backend on Pantheon? Yes, APCu can be used as a cache backend or a “key-value store”; however, this is not recommended. APCu lacks the ability to span multiple application containers. Instead, Pantheon provides a Redis-based Object Cache as a caching backend for Drupal and WordPress, which has coherence across multiple application containers. This was from [Pantheon docs](#) FAQ's:

Drupal 8 has a so-called [fast-chained backend](#) as the default cache backend, which allows to store data directly on the web server while ensuring it is correctly synchronized across multiple servers. APCu is the user cache portion of APC (Advanced PHP Cache), which has served us well till PHP 5.5 got its own zend opcache. You can think of it as a key-value store that is stored in memory and the basic operations are `apc_store($key, $data)`, `apc_fetch($keys)` and `apc_delete($keys)`. For windows the equivalent on IIS would be WinCache (<http://drupal.org/project/wincache>).

class ChainedFastBackend

Defines a backend with a fast and a consistent backend chain.

In order to mitigate a network roundtrip for each cache get operation, this cache allows a fast backend to be put in front of a slow(er) backend. Typically the fast backend will be something like APCu, and be bound to a single web node, and will not require a network round trip to fetch a cache item. The fast backend will also typically be inconsistent (will only see changes from one web node). The slower backend will be something like Mysql, Memcached or Redis, and will be used by all web nodes, thus making it consistent, but also require a network round trip for each cache get.

In addition to being useful for sites running on multiple web nodes, this backend can also be useful for sites running on a single web node where the fast backend (e.g., APCu) isn't shareable between the web and CLI processes. Single-node configurations that don't have that limitation can just use the fast cache backend directly.

We always use the fast backend when reading (`get()`) entries from cache, but check whether they were created before the last write (`set()`) to this (chained) cache backend. Those cache entries that were created before the last write are discarded, but we use their cache IDs to then read them from the consistent (slower) cache backend instead; at the same time we update the fast cache backend so that the next read will hit the faster backend again. Hence we can guarantee that the cache entries we return are all up-to-date, and maximally exploit the faster cache backend. This cache backend uses and maintains a “last write timestamp” to determine which cache entries should be discarded.

Because this backend will mark all the cache entries in a bin as out-dated for each write to a bin, it is best suited to bins with fewer changes.

Note that this is designed specifically for combining a fast inconsistent cache backend with a slower consistent cache back-end. To still function correctly, it needs to do a consistency check (see the “last write timestamp” logic). This contrasts with \Drupal\Core\Cache\BackendChain, which assumes both chained cache backends are consistent, thus a consistency check being pointless. This information is from

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Cache%21ChainedFastBackend.php/class/ChainedFastBackend/9>

APCu

APCu is the official replacement for the outdated APC extension. APC provided both opcode caching (opcache) and object caching. As PHP versions 5.5 and above include their own opcache, APC was no longer compatible, and its opcache functionality became useless. The developers of APC then created APCu, which offers only the object caching (read “in memory data caching”) functionality (they removed the outdated opcache). Read more at <https://www.php.net/manual/en/book.apcu.php>

NOTE

APCu is not the same as apc!

APCu support is built into Drupal Core. From this [Change record Sep 2014](#):

In order to improve cache performance, Drupal 8 now has:

A `cache.backend.apcu` service that site administrators can assign as the backend of a cache bin via `$settings['cache']` in `settings.php` for sites running on a single server, with a PHP installation that has APCu enabled, and that do not use Drush or other command line scripts.

WARNING

This references single-server sites not needing Drush. TODO: I couldn't find any references to using APCu with multi-server setups so I'm not sure if that is a safe configuration.

A `cache.backend.chainedfast` service that combines APCu availability detection, APCu front caching, and cross-server / cross-process consistency management via chaining to a secondary backend (either the database or whatever is configured for `$settings['cache']['default']`).

A default_backend service tag (the value of which can be set to a backend service name, such as cache.backend.chainedfast) that module developers can assign to cache bin services to identify bins that are good candidates for specialized cache backends.

The above tag assigned to the cache.bootstrap, cache.config, and cache.discovery bin services.

This means that by default (on a site with nothing set for \$settings['cache'] in settings.php), the bootstrap, config, and discovery cache bins automatically benefit from APCu caching if APCu is available, and this is compatible with Drush usage (e.g., Drush can be used to clear caches and the web process receives that cache clear) and multi-server deployments.

APCu will act as a very fast local cache for all requests. Other cache backends can act as bigger, more general cache backend that is consistent across processes or servers.

For module developers creating custom cache bins

If you are defining a cache bin that is:

- relatively small (likely to have few enough entries to fit within APCu memory), and
- high-read (many cache gets per request, so reducing traffic to the database or other networked backend is worthwhile), and
- low-write (because every write to the bin will invalidate the entire APCu cache of that bin)

then, you can add the default_backend tag to your bin, like so:

```
#example.services.yml
services:
  cache.example:
    class: Drupal\Core\Cache\CacheBackendInterface
    tags:
      - { name: cache.bin, default_backend: cache.backend.chainedfast }
    factory_method: get
    factory_service: cache_factory
    arguments: [example]
```

For site administrators customizing \$settings['cache'] Any entry for \$settings['cache']['default'] takes precedence over the default_backend service tag values, so you can disable all APCu caching by setting \$settings['cache']['default'] = 'cache.backend.database'. If you have \$settings['cache']['default'] set to some alternate backend (e.g., memcache), but would still like to benefit from APCu front caching of some bins, you can add those assignments, like so:

```
$settings['cache']['bins']['default'] = 'cache.backend.memcache';
$settings['cache']['bins']['bootstrap'] = 'cache.backend.chainedfast';
$settings['cache']['bins']['config'] = 'cache.backend.chainedfast';
$settings['cache']['bins']['discovery'] = 'cache.backend.chainedfast';
// ...
```

The bins set to use cache.backend.chainedfast will use APCu as the front cache to the default backend (e.g., memcache in the above example).

For site administrators of single-server sites that don't need Drush or other CLI access

WARNING

This references single-server sites not needing Drush. TODO: I couldn't find any references to using APCu with multi-server setups so I'm not sure if that is a safe configuration.

Pantheon docs ask in their FAQ Can APCu be used as a cache backend on Pantheon? Yes, APCu can be used as a cache backend or a "key-value store"; however, this is not recommended. APCu lacks the ability to span multiple application containers. Instead, Pantheon provides a Redis-based Object Cache as a caching backend for Drupal and WordPress, which has coherence across multiple application containers. This was from [Pantheon docs](#) FAQ's:

You can optimize further by using APCu exclusively for certain bins, like so:

```
$settings['cache']['bins']['bootstrap'] = 'cache.backend.apcu';
$settings['cache']['bins']['config'] = 'cache.backend.apcu';
$settings['cache']['bins']['discovery'] = 'cache.backend.apcu';
```

For site administrators wanting a different front cache than APCu

You can copy the cache.backend.chainedfast service definition from core.services.yml to sites/default/services.yml and add arguments to it.

For example:

```
#services.yml
services:
  cache.backend.chainedfast:
    class: Drupal\Core\Cache\ChainedFastBackendFactory
    arguments: ['@settings', 'cache.backend.eaccelerator']
    calls:
      - [setContainer, '@service_container']]
```

Reference

- Drupal: cache tags for all, regardless of your backend From Matt Glaman 22, August 2022 <https://mglaman.dev/blog/drupal-cache-tags-all-regardless-your-backend>
- Debugging your render cacheable metadata in Drupal From Matt Glaman 14, February 2023 <https://mglaman.dev/blog/debugging-your-render-cacheable-metadata-drupal>
- Cache contexts overview on drupal.org <https://www.drupal.org/docs/drupal-apis/cache-api/cache-contexts>
- Caching in Drupal 8 a quick overview of Cache tags, cache context and cache max-age with simple examples <https://zu.com/articles/caching-drupal-8>
- Nedcamp video on caching by Kelly Lucas from November 2018 https://www.youtube.com/watch?v=QCZe2K13bd0&list=PLgfWMnI57dv5KmHaK4AngrQArjoO_ylaM&t=0s&index=16
- #! code: Drupal 9: Debugging Cache Problems With The Cache Review Module, September 2022 <https://www.hashbangcode.com/article/drupal-9-debugging-cache-problems-cache-review-module>
- #! code: Drupal 9: Using The Caching API To Store Data, April 2022 <https://www.hashbangcode.com/article/drupal-9-using-caching-api-store-data>
- #! code: Drupal 8: Custom Cache Bin, September 2019 <https://www.hashbangcode.com/article/drupal-8-custom-cache-bins>
- New cache backend configuration order, per-bin default before default configuration (How to specify cache backend), June 2016 <https://www.drupal.org/node/2754947>
- [Cache API Drupal Core](#)

[Back to top](#)

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

Page last modified: May 31 2023.

[Edit this page on GitHub](#)

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

Composer, Updates and Patches

TABLE OF CONTENTS

- [Creating a local patch to a contrib module](#)
- [composer.json patches in separate file](#)
- [Stop files being overwritten during composer operations](#)
- [Updating Drupal Core](#)
- [Test composer \(dry run\)](#)
- [Version constraints](#)
- [Troubleshooting](#)
 - [Composer won't update Drupal core](#)
 - [The big reset button](#)
- [Reference](#)

views 122

Creating a local patch to a contrib module

See Making a patch at <https://www.drupal.org/node/707484>

In this case, I had the file_entity module installed and wanted to hide the tab “[files.]{.underline}” The tab item is provided by a task (read “menu tab”) in the web/modules/contrib/file_entity/file_entity.links.task.yml

```
entity.file.collection:  
  route_name: entity.file.collection  
  base_route: system.admin_content  
  title: 'Files'  
  description: 'Manage files for your site.'
```

For my patch, I want to remove this section of the file_entity.links.task.yml file.

First get the repo/git version of the module

```
$ composer update drupal/file_entity --prefer-source
```

Change the file in the text editor

Run git diff to see the changes:

```
$ git diff
```

The output shows:

```
diff --git a/file_entity.links.task.yml b/file_entity.links.task.yml
index 3ea93fc..039f7f9 100644
--- a/file_entity.links.task.yml
+++ b/file_entity.links.task.yml
@@ -15,12 +15,6 @@ entity.file.edit_form:
base_route: entity.file.canonical
weight: 0

-entity.file.collection:
- route_name: entity.file.collection
- base_route: system.admin_content
- title: 'Files'
- description: 'Manage files for your site.'

entity.file.add_form:
route_name: entity.file.add_form
base_route: entity.file.add_form
```

Create the patch

```
git diff >file_entity_disable_file_menu_tab.patch
```

Add the patch to the patches section of composer.json. Notice below the line starting with "drupal/file_entity":

```
"patches": {
  "drupal/commerce": {
    "Allow order types to have no carts": "https://www.drupal.org/files/issues/2018-03-16/commerce-direct-checkout-50.patch"
  },
  "drupal/views_load_more": {
    "Template change to keep up with core": "https://www.drupal.org/files/issues/views-load-more-pager-class-2543714-02.patch",
    "Problems with exposed filters": "https://www.drupal.org/files/issues/views_load_more-problems-with-exposed-filters-2630306-4.patch"
  },
  "drupal/easy_breadcrumb": {
    "Titles in breadcrumbs are double-escaped": "https://www.drupal.org/files/issues/2018-06-21/2979389-7-easy-breadcrumb--double-escaped-titles.patch"
  },
  "drupal/file_entity": {
    "Temporarily disable the files menu tab": "./patches/file_entity_disable_file_menu_tab.patch"
  }
}
```

Revert the file in git and then try to apply the patch.

Here is the patch command way to un-apply or revert a patch (-R means revert)

```
patch -p1 -R < ./patches/fix_scary_module.patch
```

To apply the patch:

```
patch -p1 < ./patches/fix_scary_module.patch
```

composer.json patches in separate file

To separate patches into a different file other than composer json add "patches-file" section under "extra". See example below:

```

"extra": {
  "installer-paths": {
    "web/core": ["type:drupal-core"],
    "web/libraries/{$name}": ["type:drupal-library"],
    "web/modules/contrib/{$name}": ["type:drupal-module"],
    "web/profiles/contrib/{$name}": ["type:drupal-profile"],
    "web/themes/contrib/{$name}": ["type:drupal-theme"],
    "drush/Commands/contrib/{$name}": ["type:drupal-drush"],
    "web/modules/custom/{$name}": ["type:drupal-custom-module"],
    "web/themes/custom/{$name}": ["type:drupal-custom-theme"]
  },
  "drupal-scaffold": {
    "locations": {
      "web-root": "web/"
    },
    "excludes": [
      "robots.txt",
      ".htaccess"
    ]
  },
  "patches-file": "patches/composer.patches.json"
}

```

If composer install fails, try `composer -vvv` for verbose output

If the issue is that it can't find the file for example if it displays the following:

```

- Applying patches for drupal/addtocalendar
./patches/add_to_calendar_smart_date_handling.patch (Add support for smart_date fields)
patch '-p1' --no-backup-if-mismatch -d 'web/modules/contrib/addtocalendar' < '/Users/selwyn/Sites/bxglobal/patches/add_to_calendar_smart_date_handling.patch'
Executing command (CWD): patch '-p1' --no-backup-if-mismatch -d 'web/modules/contrib/addtocalendar' < '/Users/selwyn/Sites/ttxglobal/patches/add_to_calendar_smart_date_h
can't find file to patch at input line 5
Perhaps you used the wrong -p or --strip option?

```

This means the patch is trying to run the patch in the directory `web/modules/contrib/addtocalendar` (notice the `-d web/modules/contrib/addtocalendar` above)

In this case, recreate the patch with the `--no-prefix` option i.e.

```
git diff --no-prefix >./patches/patch2.patch
```

Then composer install will apply the patch correctly

More at <https://github.com/cweagans/composer-patches/issues/146>

Stop files being overwritten during composer operations

Depending on your `composer.json`, files like `development.services.yml` may be overwritten from during scaffolding. To prevent certain scaffold files from being overwritten every time you run a Composer command you can specify them in the “extra” section of your project’s `composer.json`. See the docs on Excluding scaffold files.

The following snippet prevents the `development.services.yml` from being regularly overwritten:

```
"drupal-scaffold": {  
  "locations": {  
    "web-root": "web/"  
  },  
  "file-mapping": {  
    "[web-root]/sites/development.services.yml": false  
  }  
},
```

The code above is from <https://www.drupal.org/docs/develop/development-tools/disable-caching#s-beware-of-scaffolding>

and from https://www.drupal.org/docs/develop/using-composer/using-drupals-composer-scaffold#toc_6: Sometimes, a project might prefer to entirely replace a scaffold file provided by a dependency, and receive no further updates for it. This can be done by setting the value for the scaffold file to exclude to false. In the example below, three files are excluded from being overwritten:

```
"name": "my/project",  
...  
"extra": {  
  "drupal-scaffold": {  
    "locations": {  
      "web-root": "web/"  
    },  
    "file-mapping": {  
      "[web-root]/robots.txt": false  
      "[web-root]/.htaccess": false,  
      "[web-root]/sites/development.services.yml": false  
    },  
    ...  
  }  
}
```

More at <https://drupal.stackexchange.com/questions/290989/composer-keeps-overwriting-htaccess-and-other-files-every-time-i-do-anything>

Updating Drupal Core

if there is drupal/core-recommended in your composer.json use:

```
$ composer update drupal/core-recommended -W
```

if there is no drupal/core-recommended in your composer.json use:

```
$ composer update drupal/core -W
```

Note composer update -w is the same as composer update --with-dependencies

Test composer (dry run)

If you want to run through an installation without actually installing a package, you can use --dry-run. This will simulate the installation and show you what would happen.

```
composer update --dry-run "drupal/*"
```

produces something like:

```
Package operations: 0 installs, 4 updates, 0 removals  
- Updating drupal/core (8.8.2) to drupal/core (8.8.4)  
- Updating drupal/config_direct_save (1.0.0) to drupal/config_direct_save (1.1.0)  
- Updating drupal/core-recommended (8.8.2) to drupal/core-recommended (8.8.4)  
- Updating drupal/crop (1.5.0) to drupal/crop (2.0.0)
```

Version constraints

1. The caret constraint (^): this will allow any new versions except BREAKING ones—in other words, the first number in the version cannot increase, but the others can. drupal/foo:^1.0 would allow anything greater than or equal to 1.0 but less than 2.0.x . If you need to specify a version, this is the recommended method.
2. The tilde constraint (~): this is a bit more restrictive than the caret constraint. It means composer can download a higher version of the last digit specified only. For example, drupal/foo:~1.2 will allow anything greater than or equal to version 1.2 (i.e., 1.2.0, 1.3.0, 1.4.0, ..., 1.999.999), but it won't allow that first 1 to increment to a 2.x release. Likewise, drupal/foo:~1.2.3 will allow anything from 1.2.3 to 1.2.999, but not 1.3.0.
3. The other constraints are a little more self-explanatory. You can specify a version range with operators, a specific stability level (e.g., -stable or -dev), or even specify wildcards with *.

Version range: By using comparison operators you can specify ranges of valid versions. Valid operators are >, >=, <, <=, !=.

You can define multiple ranges. Ranges separated by a space () or comma (,) will be treated as a logical AND. A double pipe (||) will be treated as a logical OR. AND has higher precedence than OR.

Note: Be careful when using unbounded ranges as you might end up unexpectedly installing versions that break backwards compatibility. Consider using the caret operator instead for safety.

Examples:

- >=1.0
- >=1.0 <2.0
- >=1.0 <1.1 || >=1.2

More at <https://getcomposer.org/doc/articles/versions.md>

Troubleshooting

Composer won't update Drupal core

The prohibits command tells you which packages are blocking a given package from being installed. Specify a version constraint to verify whether upgrades can be performed in your project, and if not why not.

Why won't composer install Drupal version 8.9.1.

```
composer why-not drupal/core:8.9.1
```

The big reset button

If composer bars with a bunch of errors, try removing vendor, /core, modules/contrib (and optionally composer.lock) using:

```
$ rm -fr core/ modules/contrib/ vendor/
```

Then try run composer install again to see how it does:

```
$ composer install --ignore-platform-reqs
```

Note --ignore-platform-reqs is only necessary if your php on your host computer is different to the version in your DDEV containers.

You could always use this for DDEV:

```
$ ddev composer install
```

Reference

- Drupal 8 composer best practices (Jan 2018) <https://www.lullabot.com/articles/drupal-8-composer-best-practices>
- Making a patch (Dec 2022) <https://www.drupal.org/node/707484>
- Composer Documentation <https://getcomposer.org/doc/>
- Composer documentation article on versions and constraints <https://getcomposer.org/doc/articles/versions.md>

- Using Drupal's Composer Scaffold updated Dec 2922 https://www.drupal.org/docs/develop/using-composer/using-drupals-composer-scaffold#toc_6
-
-

[Back to top](#)

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

Page last modified: Apr 28 2023.

[Edit this page on GitHub](#)

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

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 yml 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 yml 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..375fb8af 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 UI 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.user 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();

\nDrupal::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'] = './configsync';
```

- drush cex -y - export entire config.

```
$ drush cex -y
[success] Configuration successfully exported to ./configsync.
./configsync
```

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 it's 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 Polit](#) 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.

CRON

TABLE OF CONTENTS

- [Overview](#)
- [How does it work?](#)
- [Enable Drupal Cron](#)
- [The cron command](#)
- [Setting up cron](#)
- [Disable Drupal cron](#)
- [hook_cron\(\)](#)
- [Common inquiries regarding cron jobs](#)
 - [When did the cron job last run?](#)
 - [How to stop Cron from continuously executing things?](#)
 - [Resolving the ip and name for cron](#)
- [Resources:](#)

views 165

Overview

Cron is a time-based task scheduler that executes commands at specified intervals, called **cron jobs**. Cron is available on Unix, Linux, and Mac servers, and Windows servers use a Scheduled Task to execute commands. Cron jobs are used in Drupal to handle maintenance tasks such as cleaning up log files and checking for updates.

How does it work?

Drupal provides an automated cron system that works with all operating systems because it does not involve the operating system's cron daemon. Instead, it works by checking at the end of each Drupal request to see when the cron last ran. If it has been too long, cron tasks are processed as part of that request.

Module `automated_cron` subscribes to the [onTerminate](#) event for request. You can read more about how this component works in Symfony [here].

File `core/modules/automated_cron/src/EventSubscriber/AutomatedCron.php`:

```
/**  
 * Registers the methods in this class that should be listeners.  
 *  
 * @return array  
 * An array of event listener definitions.  
 */  
  
public static function getSubscribedEvents(): array {  
    return [KernelEvents::TERMINATE => ['onTerminate', 100]];  
}
```

Drupal then keeps track of when the cron ran and ensures that the next time it runs is only after the configured amount of time has elapsed.

```

/**
 * Run the automated cron if enabled.
 *
 * @param \Symfony\Component\HttpKernel\Event\TerminateEvent $event
 *   The Event to process.
 */
public function onTerminate(TerminateEvent $event): void {
    $interval = $this->config->get('interval');

    if ($interval > 0) {
        $cron_next = $this->state->get('system.cron_last', 0) + $interval;

        if ((int) $event->getRequest()->server->get('REQUEST_TIME') > $cron_next) {
            $this->cron->run();
        }
    }
}

```

So, in essence, if the cron is set to run every hour but the next visitor only comes in three hours, it will only run then.

Enable Drupal Cron

- One way to enable cron is through the administration page. By default, Drupal has a built-in core automated cron system that manages cron. You can access this system by navigating to **Configuration > System > Cron** (/admin/config/system/cron). If you have just installed Drupal, this option should be enabled by default. You can confirm this by checking the status of the Automated Cron module at /admin/modules.
- Another way to enable cron is to run it manually from the **Reports > Status report page**. By default, cron runs every 3 hours, but you can change this to run every hour or every 6 hours. You can also use contributed modules for additional cron functions.
- To run cron using Drush, open a terminal or command prompt and navigate to your Drupal site's root directory. Then, enter the command drush cron. This will run cron for your site.

The cron command

To get Drupal to take care of its maintenance you should have the server execute Drupal's cron periodically. This is done by logging in to the server directly and settings the crontab file.

Crontab (CRON TABLE) - is a text file that contains the schedule of cron entries to be run at specified times. This file can be created and edited either through the command line interface.

In the following example, the crontab command shown below will activate the cron tasks automatically on the hour:

```
0 * * * * wget -O - -q -t 1 http://www.example.com/cron/<key>
```

In the above sample, the 0***** represents when the task should happen. The first figure represents minutes – in this case, on the 'zero' minute, or top of the hour. The other figures represent the hour, day, month, and day of the week. A * is a wildcard, meaning 'every time'. The minimum is every minute *****.

The rest of the line wget -O - -q -t 1 tells the server to request a URL, so the server executes the cron script.

Here is a diagram of the general crontab syntax, for illustration:

```

# +----- minute (0 - 59)
# | +----- hour (0 - 23)
# | | +----- day of the month (1 - 31)
# | | | +---- month (1 - 12)
# | | | | +-- day of the week (0 - 6) (Sunday=0)
# | | | |
* * * * * command to be executed

```

Thus, the cron command example above means ping http://www.example.com/cron/<key> at the zero minutes on every hour of every day of every month of every day of the week.

Setting up cron

To edit a crontab through the command line, type:

- 1 At the Linux command prompt, type: `sudo crontab -e`
- 2 Add ONE of the following lines:

```
45 * * * * wget -O - -q -t 1 http://www.example.com/cron/<key>
45 * * * * curl -s http://example.com/cron/<key>
```

This would have a wget or curl visit your cron page 45 minutes after every hour.

- 3 Save and exit the file. Check the Drupal status report, which shows the time of the cron execution.

NOTE

Use [crontab guru](#) - it's a quick and easy editor for cron schedule expressions.

Disable Drupal cron

For performance reasons, or if you want to ensure that cron can only ever run from an external trigger (not from Drupal), it may be desirable to disable Drupal's automated cron system, in one of three ways:

- 1 The preferred way to disable Drupal's core automated cron module is by unchecking it at </admin/modules>.
- 2 To temporarily disable cron, set the 'Run cron every' value to 'Never' (e.g., at **Administration > Configuration > System > Cron** (</admin/config/system/cron>)).
- 3 For advanced reasons, another way to disable cron in Drupal is to add the following line to your settings.php. Note that this fixes the setting at </admin/config/system/cron> to 'Never', and administrative users cannot override it.

```
$config['automated_cron.settings']['interval'] = 0;
```

hook_cron()

Gets fired every time the cron runs, so basically, Drupal's cron is a collection of function calls to various modules. For this reason, we must avoid overloading the request with heavy processing; otherwise, the request might crash.

```
function announcements_feed_cron() {
  $config = \Drupal::config('announcements_feed.settings');
  $interval = $config->get('cron_interval');
  $last_check = \Drupal::state()->get('announcements_feed.last_fetch', 0);
  $time = \Drupal::time()->getRequestTime();
  if ($time - $last_check > $interval) {
    \Drupal::service('announcements_feed.fetcher')->fetch(TRUE);
    \Drupal::state()->set('announcements_feed.last_fetch', $time);
  }
}
```

Common inquiries regarding cron jobs

When did the cron job last run?

We can use this in .module files (which don't allow dependency injection) in this way.

```
// Find out when cron was last run; the key is 'system.cron_last'.
$cron_last = \Drupal::state()->get('system.cron_last');
```

Or in another file, we need to use dependency injection.

```
$cron_last = $this->state->get('system.cron_last')
```

How to stop Cron from continuously executing things?

To stop cron from endlessly executing pending cron tasks truncate the queue table e.g. if you have queued up work such as in the salesforce module.

Resolving the ip and name for cron

Here is a Drupal cron job on a prod server where it uses a --resolve param to resolve the IP and the name. This task runs every 15 minutes.

```
*15 * * * * curl -svo /dev/null http://prod.ddd.test.gov:8080/cron/<key> --resolve prod.ddd.test.gov:8080:201.86.28.12
```

Resources:

- [Link 1](#)
- [Link 2](#)

[Back to top](#)

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

Page last modified: May 2 2023.

[Edit this page on GitHub](#)

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

Dates and Times

TABLE OF CONTENTS

- [Overview](#)
- [Retrieve a date field](#)
- [Retrieve date range field](#)
- [Formatting date range fields](#)
- [Formatting a date string with an embedded timezone](#)
- [Formatting a date range for display](#)
- [Saving date fields](#)
- [Create DrupalDateTime objects](#)
 - [Create DrupalDateTime objects with timezones](#)
 - [Create a DrupalDateTime object and display as a year only](#)
 - [Formatting node created time with Drupal date.formatter service](#)
 - [Date arithmetic example 1](#)
 - [Date arithmetic example 2](#)
 - [Comparing DrupalDateTime values](#)
 - [Comparing dates \(without comparing times\)](#)
 - [Comparing Dates to see if a node has expired](#)
 - [Node creation and changed dates](#)
 - [Query the creation date using entityQuery](#)
 - [Query a date field with no time](#)
 - [Query a date field with a time](#)
 - [Smart Date](#)
 - [Smart date: Load and format](#)
 - [Smart date: all-day](#)
 - [Smart date: Range of values](#)
- [Reference](#)
 - [Date field storage](#)
 - [DrupalDateTime API reference](#)
 - [UTC](#)
 - [Unix epoch timestamps](#)
 - [Links](#)

views 106

Overview

Drupal Date fields are stored as varchar 20 UTC date strings (e.g. 2022-06-30T12:00:00) while node created and changed fields are stored as int 11 containing Unix epoch timestamps (e.g. 1656379475) in the node_field_data table (fields: created and changed).

Accessing date fields comes in many flavors:

```

// Returns 2021-12-27 for a date only field.
$event_date = $event_node->field_event_date->value;

// Returns 2021-12-28T16:00:00 for a date field with time.
$event_datetime = $event_node->field_event_datetime->value;

// Return a Unix epoch timestamp.
$timestamp = $event_node->field_date->date->getTimestamp();

// Return a formatted date string.
$date_formatted = $event_node->field_date->date->format('Y-m-d H:i:s');

```

Using `$node->field_mydatefield->date` is ideal as it returns a `DrupalDateTime` class which gives you all sorts of goodness including date math capabilities and formatting.

If you need to do calculations involving Unix timestamps, then using `$node->field_mydatefield->getTimestamp()` is useful although `DrupalDateTime` is probably better. More about `DrupalDateTime` at
<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>. Also at
<https://drupal.stackexchange.com/questions/252333/how-to-get-formatted-date-string-from-a-datetimeitem-object>

See [Nodes and Fields chapter Date fields section](#) for more on date fields

Retrieve a date field

You can retrieve date fields a few different ways. They are stored as varchar 20 UTC date strings e.g. 2022-06-30T12:00:00

```

// For a date only field, this returns a string like: 2024-08-31.
// For a date field with time, this returns: 2021-12-28T16:00:00.
$end_date = $contract_node->field_contract_date->value;

// Returns unix timestamp e.g. 1725105600
$end_date = $contract_node->field_contract_date->date->getTimestamp();

// Returns a DrupalDateTime object with all its goodness which you can format.
$end_date = $contract_node->field_contract_date->date;
$formatted_date = $end_date->format('m/d/y');

```

Retrieve date range field

To retrieve a date range field from a node, use `value` and `end_value` for the stand and end dates:

```

// Magic getters.
$start = $event_node->field_event_date_range->value
$end = $event_node->field_event_date_range->end_value

// Using get().
$start = $event_node->get('field_event_date_range')->value
$end = $event_node->get('field_event_date_range')->end_value

// Using getValue().
$start = $event_node->get('field_event_date_range')->getValue()[0]['value'];
$end = $event_node->get('field_event_date_range')->getValue()[0]['end_value'];

```

Formatting date range fields

Here are two different examples for formatting date fields:

```
// formatted start date
$start_date_formatted = $node->field_date->start_date->format('Y-m-d H:i:s');
// formatted end date
$end_date_formatted = $node->field_date->end_date->format('Y-m-d H:i:s');
```

Use this link at [php.net](https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-format,-character) for date format strings <https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-format,-character>

Formatting a date string with an embedded timezone

Here a date string with an embedded timezone is used to create a `DrupalDateTime` object which is then converted to be stored into a node.

```
use Drupal\Core\Datetime\DrupalDateTime;

$date_string = "2020-08-24T15:28:04+00:00";
$dtt = new DrupalDateTime($date_string);
$newstring = $dtt->format("Y-m-d\Th:i:s");
$node->set('field_date', $newstring);
```

Formatting a date range for display

This code shows how to load a date range field from a node. It will ordinarily display like 3/30/2019 - 3/31/2023 however we want it to display like Mar 30-31, 2023.

First we retrieve the starting and ending value like this:

```
$from = $node->get('field_date')->getValue()[0]['value'];
$to = $node->get('field_date')->getValue()[0]['end_value'];
```

Here is the entire function as implemented as a `hook_preprocess_node` function in a `.theme` file. We are creating a `scrunch_date` variable to be rendered via a Twig template as shown below:

```

use Drupal\Core\Datetime\DrupalDateTime;

/**
 * Implements hook preprocess_node
 *
 * @param $variables
 */
function vst_preprocess_node(&$variables) {
  if (!empty($variables['content']['field_date'])) {
    $date = $variables['content']['field_date'];

    $from = new DrupalDateTime($variables['node']->get('field_date')->getValue()[0]['value']);
    $date_array = explode("-", $from);
    $from_day = substr($date_array[2], 0, 2);
    $from_month = $date_array[1];

    $to = new DrupalDateTime($variables['node']->get('field_date')->getValue()[0]['end_value']);
    $date_array = explode(".", $to);
    $to_day = substr($date_array[2], 0, 2);
    $to_month = $date_array[1];

    if ($from_month === $to_month && $from_day != $to_day) {
      $variables['scrunch_date'] = [
        '#type' => 'markup',
        '#markup' => $from->format("M j-") . $to->format("j, Y"),
      ];
    }
  }

  // For debugging
  // kint($variables);
  // or
  // kpm($variables);
}

```

Now in the twig node template we can output the `scrunch_date` we created.

From `/web/themes/verygood/templates/node/node--seminar--teaser.html.twig`.

```

{% if content.field_date %}
  {% if scrunch_date %}
    <div>
      {{ scrunch_date }}
    </div>
  {% else %}
    <div>
      {{ content.field_date }}
    </div>
  {% endif %}
  {% endif %}

```

Saving date fields

Date fields in Drupal are stored as UTC date strings (e.g. 2022-06-30T12:00:00) and when you use `get()` or `set()`, they return strings. If you want to manipulate them, convert them to `DrupalDateTime` objects, then convert them back to strings for saving.

```
$node->set('field_date', '2025-12-31');
$node->set('field_datetime', '2025-12-31T23:59:59');
// Use this for storing created and changed.
$node->set('created', '1760140799');
$node->save();
```

Create DrupalDateTime objects

```
use Drupal\Core\Datetime\DrupalDateTime;

$date = DrupalDateTime::createFromFormat('j-M-Y', '20-Jul-2022');

// Use current date and time
$date = new DrupalDateTime('now');
// Format like Tue, Jul 16, 2022 - 11:34:am
print $date->format('l, F j, Y - H:i');

// OR
// Format like 16-07-2022: 11:43 AM
print $date->format('d-m-Y: H:i A');
```

Create DrupalDateTime objects with timezones

```
// Use current date & time.
$date = new DrupalDateTime();
$date->setTimezone(new \DateTimeZone('America/Chicago'));
// Print current time for the given time zone e.g. 01/23/2023 10:00 pm
print $date->format('m/d/Y g:i a');

// Another variation using specific date and UTC zone
$date = new DrupalDateTime('2019-07-31 11:30:00', 'UTC');
$date->setTimezone(new \DateTimeZone('America/Chicago'));
// prints 07/31/2019 6:30 am
print $date->format('m/d/Y g:i a');
```

UTC: https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Nice article on writing date fields programmatically with more info on UTC timezone at [https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically,\)%3B%20%2F%20For%20datetime%20fields](https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically,)%3B%20%2F%20For%20datetime%20fields).

Create a DrupalDateTime object and display as a year only

This code creates a `Drupal\Core\Datetime\DrupalDateTime` object and returns the year in a render array with some markup. `DrupalDateTimes` are derived from `DateTimePlus` which is a wrapper for PHP `DateTime` class.

```
use Drupal\Core\Datetime\DrupalDateTime;

public function build() {
  $date = new DrupalDateTime();
  return [
    '#markup' => t('Copyright @year&copy; My Company', [
      '@year' => $date->format('Y'),
    ]),
  ];
}
```

Formatting node created time with Drupal date.formatter service

If you want to use a custom date format for your created node date/time you can use one of the methods shown below:

```

$created_date = $node->getCreatedTime();

// Displays 05/04/2022 3:49 pm
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date, 'custom', 'm/d/Y g:i a');

// Displays 2022-05-04 15:49:30
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date, 'custom', 'Y-m-d H:i:s');

// Displays Wed, 05/04/2022 - 15:49
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date);

// Create a DrupalDateTime object and use ->format()
$created_date = $event_node->getCreatedTime();
$cdt = DrupalDateTime::createFromTimestamp($created_date);
$formatted_created_date = $cdt->format('m/d/Y g:i a');

```

See PHP Date format strings: <https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-.format,-character>

Date arithmetic example 1

The code below shows how to add \$days (an integer) to the date value retrieved from the field: field_cn_start_date and save that to the field field_cn_end_date.

```

use Drupal\Core\Datetime\DrupalDateTime;

$start_date_val = $node->get('field_cn_start_date')->value;
$days = intval($node->get('field_cn_suspension_length')->value) - 1;
$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val );
$end_date->modify("+{$days} days");
$end_date = $end_date->format("Y-m-d");

$node->set('field_cn_end_date', $end_date);
$node->save();

```

Date arithmetic example 2

Here is an example from a module showing a hook_entity_type_presave() where some data is changed as the node is being saved. The date arithmetic is pretty simple but the rest of the code is kinda messy.

This is the date arithmetic part:

```

$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
$end_date->modify("+{$days} days");
$end_date = $end_date->format("Y-m-d");
$node->set('field_cn_end_date', $end_date);

```

The rest of this code does some convoluted wrangling to figure out end dates based on user permissions, changes a node title, looks to see if this is an extension of a previously submitted notice and grabs some date fields from the original notice for use those in the current node.

```

/**
 * Implements hook_ENTITY_TYPE_presave().
 */
function ogg_mods_node_presave(NodeInterface $node) {
  switch ($node->getType()) {
    case 'cat_notice':
      $end_date = NULL != $node->get('field_cn_start_end_dates')->end_value ? $node->get('field_cn_start_end_dates')->end_value : 'n/a';
      $govt_body = NULL != $node->field_cn_governmental_body->value ? $node->field_cn_governmental_body->value : 'Unnamed Government Body';
      $start_date_val = $node->get('field_cn_start_date')->value;

      $accountProxy = \Drupal::currentUser();
      $account = $accountProxy->getAccount();
      // Anonymous users automatically fill out the end_date.
      if (!$account->hasPermission('administer cat notice')) {
        $days = intval($node->get('field_cn_suspension_length')->value) - 1;

        $end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
        $end_date->modify("+$days days");
        $end_date = $end_date->format("Y-m-d");
        $node->set('field_cn_end_date', $end_date);
      }

      // Always reset the title.
      $title = substr($govt_body, 0, 200) . " - $start_date_val";
      $node->setTitle($title);

    /*
     * Fill in Initial start and end dates if this is an extension of
     * a previously submitted notice.
     */
      $extension = $node->get('field_cn_extension')->value;
      if ($extension) {
        $previous_notice_nid = $node->get('field_cn_original_notice')->target_id;
        $previous_notice = Node::load($previous_notice_nid);
        if ($previous_notice) {
          $initial_start = $previous_notice->get('field_cn_start_date')->value;
          $initial_end = $previous_notice->get('field_cn_end_date')->value;
          $node->set('field_cn_initial_start_date', $initial_start);
          $node->set('field_cn_initial_end_date', $initial_end);
        }
      }

      break;
  }
}

```

Comparing DrupalDateTime values

The `DrupalDateTime` class extends the `DateTimePlus` class which is a wrapper for PHP `DateTime` class. That functionality allows you to do comparisons. It is probably better manners to use `DrupalDateTime` instead of `DateTime` but here is some `DateTime` code showing how to compare DateTimes.

```

date_default_timezone_set('Europe/London');

$d1 = new DateTime('2008-08-03 14:52:10');
$d2 = new DateTime('2008-01-03 11:11:10');
var_dump($d1 == $d2);
var_dump($d1 > $d2);
var_dump($d1 < $d2);

// Returns.
bool(false)
bool(true)
bool(false)

```

Comparing dates (without comparing times)

Use the setTime() function to remove the time part of a datetime so we can make comparisons of just the date.

From a form validation in a .module file.

```

function ogg_mods_cn_form_validate($form, FormStateInterface $form_state) {
  $start_date = $form_state->getValue('field_cn_start_date');
  if ($start_date) {
    $start_date = $start_date[0]['value'];
    $start_date->setTime(0, 0, 0);
    $now = new Drupal\Core\Datetime\DrupalDateTime();
    // Subtract 2 days.
    $now->modify("-2 days");
    // Clear the time.
    $now->setTime(0, 0, 0);

    \Drupal::messenger()->addMessage("Start date = $start_date");
    \Drupal::messenger()->addMessage("Now date - 2 days = $now");

    if ($start_date < $now) {
      $form_state->setErrorByName('edit-field-cn-start-date-0-value-date', t('The starting date is more than 2 days in the past. Please select a later date'));
    }
  }
}

```

Comparing Dates to see if a node has expired

This code is used to check if the value in the field field_expiration_date has passed. The field_expiration_date is a standard Drupal date field in a node. In this example, the client wanted to be able to specify the expiration date for nodes.

```

$source_node = $node_storage->load($nid);
$expiration_date = $source_node->field_expiration_date->value;

// Use expiration date to un-publish expired reseller nodes to hide them.
$status = 1;
if ($expiration_date) {
  $expirationDate = DrupalDateTime::createFromFormat('Y-m-d', $expiration_date);
  $now = new DrupalDateTime();
  if ($expiration_date < $now) {
    // When expired, unpublish the node.
    $node->set('status', 0);
    $node->save(); }
}

```

It might be interesting to factor in the timezone as date fields are stored in UTC. See
https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Node creation and changed dates

Here is a function which does an `entityQuery` for a node and returns a formatted string version of the creation date. Both `created` and `changed` are stored as Unix epoch timestamps in the `node_field_data` table (int 11).

Note. This will handle epoch dates before 1970. You can also use `$node->get('changed')` to retrieve the changed date.

```
use Drupal\Core\Datetime\DrupalDateTime;

protected function loadFirstOpinionYear($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') // or DESC
    ->range(0, 1);
  $nids = $query->execute();
  if ($nids) {
    $node = $storage->load(reset($nids));
  }
  $time = $node->get('created')->value;
  $d = DrupalDateTime::createFromTimestamp($time);
  $str = $d->format('Y-m-d H:i:s');
  return $str;
}
```

To find any nodes that were changed in the last 7 days, use the following:

```
$query = \Drupal::entityQuery('node')
->condition('field_tks_program_status', 'ready_for_release')
->accessCheck(FALSE)
->condition('type', 'teks_pub_program');
// Query changed date within 7 days of today.
$query->condition('changed', strtotime('-7 days'), '>=');

$program_nids = $query->execute();
$program_nids = array_values($program_nids);
```

Query the creation date using entityQuery

This controller example queries for any events for year 2022 with a matching taxonomy term id of 5.

```

public function test2() {

  $str = "Results";
  $year = '2022';
  $term_id = 5;
  $titles = $this->eventsForYear($year, $term_id);
  $count = count($titles);
  $str .= "<br/>Found $count titles for query $year, term_id $term_id";

  foreach ($titles as $title) {
    $str .= "<br/>$title";
  }

  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $render_array;
}

private function eventsForYear($year, $term_id): array {
  // Build valid range of start dates/times.
  $format = 'Y-m-d H:i';
  $start_date = DrupalDateTime::createFromFormat($format, $year . "-01-01 00:00");
  $end_date = DrupalDateTime::createFromFormat($format, $year . "-12-31 23:59");

  $start_date_ts = $start_date->getTimestamp();
  $end_date_ts = $end_date->getTimestamp();

  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'event')
    ->condition('field_event_category', $term_id, '!=')
    ->condition('created', $start_date_ts, '>=')
    ->condition('created', $end_date_ts, '<=')
    ->sort('title', 'DESC');
  $nids = $query->execute();
  $titles = [];
  if ($nids) {
    foreach ($nids as $nid) {
      $node = Node::load($nid);
      $titles[] = $node->getTitle();
    }
  }
  return $titles;
}

```

Read more in the article: Date (range) fields and Entity Query from February 2018 at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>

and

this Stack exchange question at <https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

Query a date field with no time

Drupal date fields can be configured to store the date and time or only the date. Looking in the database, you might notice that a date-only field is stored like: 2021-12-27 which means you can query just using a string. It probably is wiser to use DrupalDateTime like this:

```

use Drupal\Core\Datetime\DrupalDateTime;
use Drupal\datetime\Plugin\Field\FieldType\DateTimelitemInterface;

public function test4() {

  // Date-only fields are stored in the database like: '2021-12-27';

  // Get a date string suitable for use with entity query.
  $date = DrupalDateTime::createFromFormat('j-M-Y', '27-Dec-2021');
  $date->setTimezone(new \DateTimeZone(DateTimelitemInterface::STORAGE_TIMEZONE));
  // NB. Specify the date-only storage format - not the datetime storage format!
  $query_date = $date->format(DateTimelitemInterface::DATE_STORAGE_FORMAT);

  // Query using =.
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'event')
    ->condition('status', 1)
    ->condition('field_event_date.value', $query_date, '=')
    ->sort('title', 'ASC');
  $nids = $query->execute();
  $count = count($nids);

  $str = "Results";
  $str .= "<br/>Found $count events for field_event_date = $query_date";
  foreach ($nids as $nid) {
    $event_node = Node::load($nid);
    $title = $event_node->getTitle();
    $date = $event_node->field_event_date->value;
    $str .= "<br/>$title - date: $date";
  }
  $str .= "<br/>";

  // Query using >.
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'event')
    ->condition('status', 1)
    ->condition('field_event_date.value', $query_date, '>')
    ->sort('title', 'ASC');
  $nids = $query->execute();
  $count = count($nids);

  $str .= "<br/>Found $count events for field_event_date > $query_date";

  foreach ($nids as $nid) {
    $event_node = Node::load($nid);
    $title = $event_node->getTitle();
    $date = $event_node->field_event_date->value;
    $str .= "<br/>$title - date: $date";
  }

  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
  return $render_array;
}

```

```

use Drupal\Core\Datetime\DrupalDateTime;
use Drupal\datetime\Plugin\Field\FieldType\DateTimelitemInterface;

/*
 * Query a date field with a time.
 */
public function test3() {

  // Get a date string suitable for use with entity query.
  $date = new DrupalDateTime();
  // This is a date/time from my local timezone.
  $date = DrupalDateTime::createFromFormat('d-m-Y H:i A', '28-12-2021 10:00 AM');
  $date->setTimezone(new \DateTimeZone(DateTimelitemInterface::STORAGE_TIMEZONE));
  $query_date = $date->format(DateTimelitemInterface::DATETIME_STORAGE_FORMAT);

  $str = "Results";

  $query = \Drupal::entityQuery('node')
    ->condition('type', 'event')
    ->condition('status', 1)
    ->condition('field_event_datetime.value', $query_date, '=')
    ->sort('title', 'ASC');
  $nids = $query->execute();
  $count = count($nids);

  $print_date = $date->format('d-m-Y H:i A');
  $str .= "<br/><strong>$count event(s) for field_event_datetime = $print_date (UTC)</strong> ";
  $date->setTimezone(new \DateTimeZone('America/Chicago'));
  $print_date = $date->format('d-m-Y H:i A');
  $str .= "<br/>For my timezone (America/Chicago), that is $print_date";

  foreach ($nids as $nid) {
    $event_node = Node::load($nid);
    $title = $event_node->getTitle();
    $display_date = $event_node->field_event_datetime->value;
    $str .= "<br/>$title - date: $display_date";
  }
  $str .= "<br/>";

  $query = \Drupal::entityQuery('node')
    ->condition('type', 'event')
    ->condition('status', 1)
    ->condition('field_event_datetime.value', $query_date, '>')
    ->sort('title', 'ASC');
  $nids = $query->execute();
  $count = count($nids);

  $print_date = $date->format('d-m-Y H:i A');
  $str .= "<br/><strong>$count event(s) for field_event_datetime > $print_date (UTC)</strong> ";
  $date->setTimezone(new \DateTimeZone('America/Chicago'));
  $print_date = $date->format('d-m-Y H:i A');
  $str .= "<br/>For my timezone (America/Chicago), that is $print_date";

  foreach ($nids as $nid) {
    $event_node = Node::load($nid);
    $title = $event_node->getTitle();
    $display_date = $event_node->field_event_datetime->value;
    $str .= "<br/>$title - date: $display_date";
  }
}

```

```

$str .= "<br/>";

$render_array['content'] = [
  '#type' => 'item',
  '#markup' => $str,
];
return $render_array;
}

```

More in the article: Date (range) fields and Entity Query from February 2018 at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>

and

this Stack exchange question at <https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

Smart Date

This module provides the date field that fills in all the gaps in functionality that Drupal core dates lack. Maybe someday it will make it into Drupal core.

This module attempts to provide a more user-friendly date field, by upgrading the functionality of core in several ways:

Easy Admin UI: Includes the concept of duration, so that a field can have a configurable default duration (e.g. 1 hour) and the end time will be auto-populated based on the start. The overall goal is to provide a smart interface for time range/event data entry, more inline with calendar applications which editors will be familiar with.

All Day Events Most calendar applications provide a one-click option to make a an event, appointment, or other time-related content span a full day. This module brings that same capability to Drupal.

Zero Duration Events Show only a single time for events that don't need a duration.

Formatting: More sophisticated output formatting, for example to show the times as a range but with a single output of the date. In the settings a site builder can control how date the ranges will be output, at a very granular level.

Performance: Dates are stored as timestamps to improve performance, especially when filtering or sorting. Concerns with the performance of core's date range have been documented in [#3048072: Date Range field creates very slow queries in Views](#).

Overall, the approach in this module is to leverage core's existing Datetime functionality, using the timestamp storage capability also in core, with some custom Javascript to add intelligence to the admin interface, and a suite of options to ensure dates can be formatted to suit any site's needs.

Display configuration is managed through translatable Smart Date Formats, so your detailed display setup is easily portable between fields, views, and so on. (From https://www.drupal.org/project/smart_date)

Smart date: Load and format

Load the smart date field and use the Drupal date formatting service (date.formatter). Smart date fields are always stored as unix timestamp values e.g. 1608566400 which need conversion for human consumption.

```

$start = $node->field_when->value;
$formatter = \Drupal::service('date.formatter');

//returns something like 12/21/2020 10:00 am
$start_time = $formatter->format($start, 'custom', 'm/d/Y g:i:a');

```

Alternatively, you could load it, create a DrupalDateTime and then format it:

```

$start = $node->field_when->value;
$dt = DrupalDateTime::createFromTimestamp($start);
$start_date = $dt->format('m/d/Y'); //returns 12/21/22
$start_time = $dt->format('g:i:a'); // returns 10:00am

```

Smart date: all-day

To check if a smart date is set to all day, check the duration. If it is 1439, that means all day.

```

$start_ts = $node->field_when->value;
$start_dt = DrupalDateTime::createFromTimestamp($start_ts);
$start_date = $start_dt->format('m/d/Y');
$duration = $node->field_when->duration; //1439 = all day
if ($duration == 1439) {
  $start_time = "all day";
}
else {
  $start_time = $start_dt->format('g:i:a');
}

```

Smart date: Range of values

```

//Event start date.

//returns a SmartDateFieldItemList
$whens = $node->get('field_when');

// Each $when is a \Drupal\smart_date\Plugin\Field\FieldType\SmartDateItem.
foreach ($whens as $when) {
  $start = $when->value;
  $end = $when->end_value;
  $duration = $when->duration; //1439 = all day
  $tz = $when->timezone; //"" means default. Uses America/Chicago type format.
}

```

You can also peek into the repeating rule and repeating rule index. These are in the `smart_date_rule` table and I believe the `index` column identifies which item is in the “instances” column.

```

$rrule = $when->rrule;
$rrule_index = $when->rrule_index;

```

Reference

Date field storage

Note. The node `created` and `changed` fields (int 11) use a Unix epoch timestamp stored in the `node_field_data` table. These have values like 1525302749. Drupal date fields (with times) are stored as UTC strings in varchar 20 fields which look like 2019-05-15T21:32:00.

DrupalDateTime API reference

The `DrupalDateTime` class extends the `DateTimePlus` class which is a wrapper for PHP `DateTime` class. It extends the basic component and adds in Drupal-specific handling, like translation of the `format()` method.

`DateTimePlus` has some static methods to create `DrupalDateTime` objects e.g.

```
DrupalDateTime::createFromArray(['year' => 2010, 'month' => 9, 'day' => 28]);
```

From <https://git.drupalcode.org/project/drupal/-/blob/10.1.x/core/lib/Drupal/Component/Datetime/DateTimePlus.php>:

```

/**
 * Wraps DateTime().
 *
 * This class wraps the PHP DateTime class with more flexible initialization
 * parameters, allowing a date to be created from an existing date object,
 * a timestamp, a string with an unknown format, a string with a known
 * format, or an array of date parts. It also adds an errors array
 * and a __toString() method to the date object.
 *
 * This class is less lenient than the DateTime class. It changes
 * the default behavior for handling date values like '2011-00-00'.
 * The DateTime class would convert that value to '2010-11-30' and report
 * a warning but not an error. This extension treats that as an error.
 *
 * As with the DateTime class, a date object may be created even if it has
 * errors. It has an errors array attached to it that explains what the
 * errors are. This is less disruptive than allowing datetime exceptions
 * to abort processing. The calling script can decide what to do about
 * errors using hasErrors() and getErrors().
 *
 * @method $this add(\DateInterval $interval)
 * @method static array getLastErrors()
 * @method $this modify(string $modify)
 * @method $this setDate(int $year, int $month, int $day)
 * @method $this setISODate(int $year, int $week, int $day = 1)
 * @method $this setTime(int $hour, int $minute, int $second = 0, int $microseconds = 0)
 * @method $this setTimestamp(int $unixtimestamp)
 * @method $this setTimezone(\DateTimeZone $timezone)
 * @method $this sub(\DateInterval $interval)
 * @method int getOffset()
 * @method int getTimestamp()
 * @method \DateTimeZone getTimezone()
 */

```

(More at <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>)

UTC

Coordinated Universal Time or UTC is the primary time standard by which the world regulates clocks and time. It is within about 1 second of mean solar time at 0° longitude (at the IERS Reference Meridian as the currently used prime meridian) such as UT1 and is not adjusted for daylight saving time. It is effectively a successor to Greenwich Mean Time (GMT). From [From https://en.wikipedia.org/wiki/Coordinated_Universal_Time](https://en.wikipedia.org/wiki/Coordinated_Universal_Time)

Unix epoch timestamps

From <https://www.unixtimestamp.com/> - The unix time stamp is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. It should also be pointed out (thanks to the comments from visitors to this site) that this point in time technically does not change no matter where you are located on the globe. This is very useful to computer systems for tracking and sorting dated information in dynamic and distributed applications both online and client side.

Links

- Drupal API DrupalDateTime Class
<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>
- From php.net, the definitive documentation on date format strings
<https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string,-format,-character>
- Coordinated Universal Time or UTC Wikipedia article https://en.wikipedia.org/wiki/Coordinated_Universal_Time

- Goran Nikolovski's article: Set date programatically from January 2019 [https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically.\)%3B%20%2F%20For%20datetime%20fields](https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically.)%3B%20%2F%20For%20datetime%20fields)
 - Patrick's article: Date (range) fields and Entity Query from February 2018 <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>
 - Drupal APIS <https://www.drupal.org/docs/drupal-apis>
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) 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.

Debugging

TABLE OF CONTENTS

- [Overview](#)
- [Enable error reporting](#)
- [Disable caches and enable Twig debugging](#)
- [Enable/Disable Xdebug](#)
- [Xdebug Port](#)
- [Drupal code debugging](#)
- [Command line or drush debugging](#)
- [Add a breakpoint in code](#)
- [Troubleshooting Xdebug with DDEV](#)
 - [Could not connect to debugging client](#)
 - [PhpStorm refuses to debug](#)
 - [Curl](#)
 - [Logs](#)
 - [Telnet](#)
 - [Is Xdebug enabled?](#)
 - [What is listening on the debug port?](#)
- [Enable twig debugging output in source](#)
- [Devel and Devel Kint Extras](#)
 - [Setup](#)
 - [Add kint to a custom module](#)
 - [Dump variables in a TWIG template](#)
 - [Kint::dump](#)
 - [Set max levels to avoid running out of memory](#)
- [Resources](#)

views 140

Overview

Using a combination of PhpStorm, DDEV and Xdebug makes debugging a pleasure. PhpStorm is not essential. Xdebug works fine with other IDE's also. In my experience, many Drupal developers have not experienced using a true debugger, but once they do, they wonder how they ever delivered any code without it.

Enable error reporting

If you experience a WSOD (White Screen Of Death), enabling verbose error messages can often give you some useful clue:

```
/** 
 * Show all error messages, with backtrace information.
 *
 * In case the error level could not be fetched from the database, as for
 * example the database connection failed, we rely only on this value.
 */
$config['system.logging']['error_level'] = 'verbose';
```

For more, check out <https://drupal.stackexchange.com/questions/127182/how-do-i-enable-developer-debug-mode#:~:text=%24config%5B'system..can%20always%20comment%20them%20out>. Also in <https://github.com/drupal/drupal/blob/10.1.x/core/includes/bootstrap.inc> Error reporting levels are defined:

```

/**
 * Error reporting level: display no errors.
 */
const ERROR_REPORTING_HIDE = 'hide';

/**
 * Error reporting level: display errors and warnings.
 */
const ERROR_REPORTING_DISPLAY_SOME = 'some';

/**
 * Error reporting level: display all messages.
 */
const ERROR_REPORTING_DISPLAY_ALL = 'all';

/**
 * Error reporting level: display all messages, plus backtrace information.
 */
const ERROR_REPORTING_DISPLAY_VERBOSE = 'verbose';

```

Disable caches and enable Twig debugging

This will cause twig debugging information to be displayed in the HTML code like the following:

```

<!-- THEME DEBUG -->
<!-- THEME HOOK: 'toolbar' -->
<!-- BEGIN OUTPUT from 'core/themes/stable/templates/navigation/toolbar.html.twig' -->

```

and

```

<!-- THEME DEBUG -->
<!-- THEME HOOK: 'page' -->
<!-- FILE NAME SUGGESTIONS:
    * page--teks--admin--srp--program--expectation--correlation--vote-all.html.twig
    * page--teks--admin--srp--program--expectation--correlation--852136.html.twig
    * page--teks--admin--srp--program--expectation--correlation--%.html.twig
    * page--teks--admin--srp--program--expectation--correlation.html.twig
    * page--teks--admin--srp--program--expectation--852131.html.twig
    * page--teks--admin--srp--program--expectation--%.html.twig
    * page--teks--admin--srp--program--expectation.html.twig
    * page--teks--admin--srp--program--852061.html.twig
    * page--teks--admin--srp--program--%.html.twig
    * page--teks--admin--srp--program.html.twig
    * page--teks--admin--srp.html.twig
    x page--teks--admin.html.twig
    * page--teks.html.twig
    * page.html.twig
-->

```

In `sites/default/development.services.yml` in the parameters, `twig.config`, set `debug: true`. See `core.services.yml` for lots of other items to change for development.

```
# Local development services.  
#  
parameters:  
  http.response.debug_cacheability_headers: true  
  twig.config:  
    debug: true  
    auto_reload: true  
    cache: false  
  
  # To disable caching, you need this and a few other items  
  services:  
    cache.backend.null:  
      class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in `settings.local.php`:

```
/**  
 * Enable local development services.  
 */  
$settings['container_yamls'][0] = DRUPAL_ROOT . '/sites/development.services.yml';
```

Disable caches in `settings.local.php`:

```
$config['system.performance']['css']['preprocess'] = FALSE;  
$config['system.performance']['js']['preprocess'] = FALSE;  
$settings['cache']['bins']['render'] = 'cache.backend.null';  
$settings['cache']['bins']['page'] = 'cache.backend.null';  
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
```

Enable/Disable Xdebug

To enable or disable Xdebug when using DDEV use:

```
$ ddev xdebug on  
  
$ ddev xdebug off
```

Note. Enabling Xdebug will slow down your app because xdebug has a significant performance impact so be sure to disable it when you are finished debugging.

Add this to your `.zshrc` or `.bash` file for `xon` and `xoff` shortcut

```
alias xon='ddev xdebug on'  
alias xoff='ddev xdebug off'
```

Xdebug Port

DDEV now sets the default port to 9003. If you want to change it to another port, use the steps below. From

<https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/#using-xdebug-on-a-port-other-than-the-default-9003>

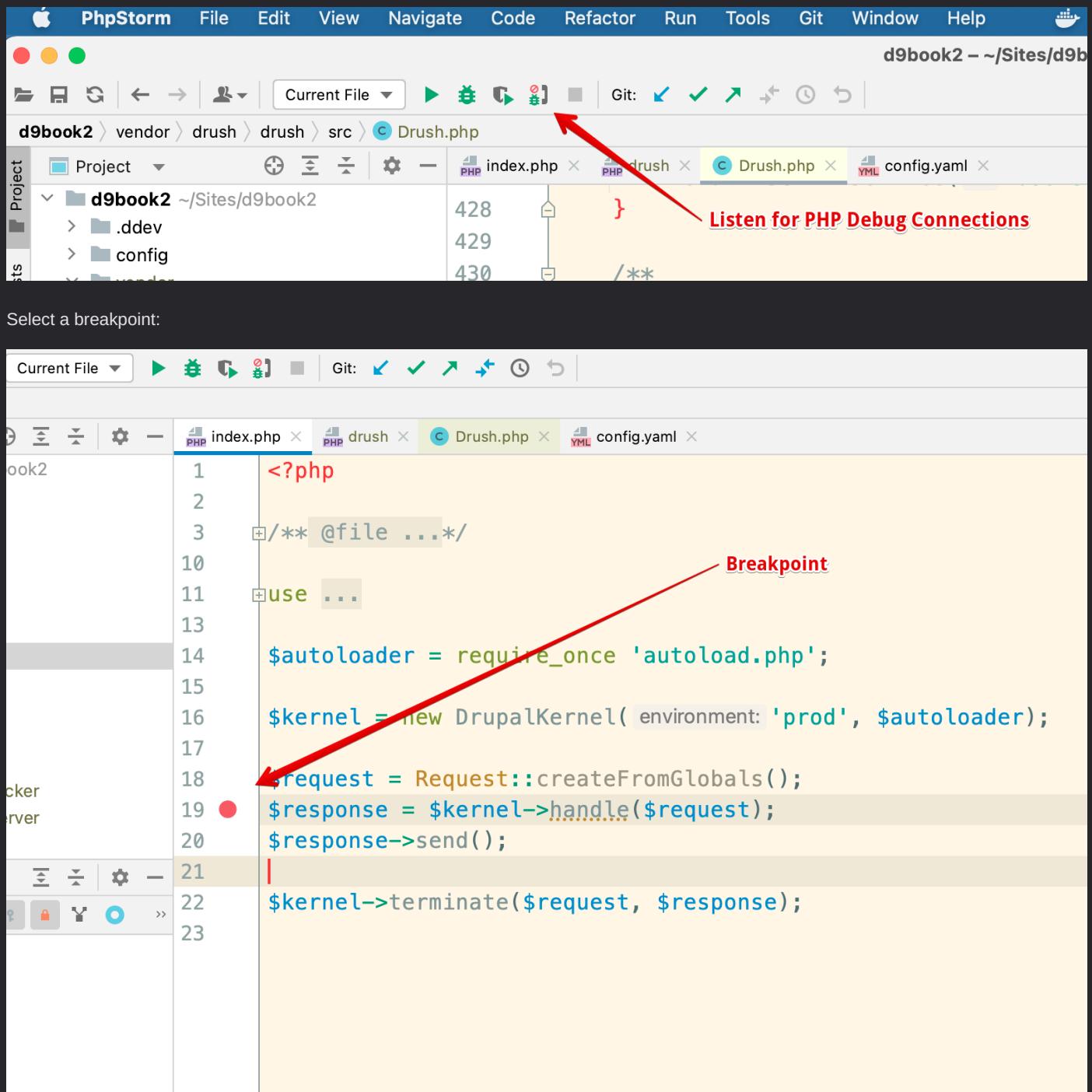
To override the port, add an override file in the project's `.ddev/php` directory. For example, add the file `.ddev/php/xdebug_client_port.ini` like this to specify the legacy port 9000:

```
[PHP]  
  
xdebug.client_port=9000
```

Drupal code debugging

Phpstorm and DDEV make this process as painless as possible. Once you enable Xdebug in DDEV, simply click the "start listening for PHP Debug Connections" button.

To start debugging, open the index.php file and set a breakpoint by clicking on a line number.



Next refresh the Drupal home page in a browser

You should immediately see a dialog pop up in PhpStorm asking you to configure your local path. Be sure to click the `site/web/index.php` and click **Accept**:

Note. If you select one of the other lines you will see a different php file pop up and you won't be debugging Drupal, but probably some Symfony file.

Incoming Connection From Xdebug

Server name: d9book2.ddev.site

Server port: 80

Request uri: /admin/reports/status

File path on server: /var/www/html/web/index.php

Configure local file path

Import mappings from deployment Manually choose local file or project

Select a project or a file to debug

d9book2

-  /Users/selwyn/Sites/d9book2/vendor/symfony/http-client-contracts/Test/Fixtures/web/index.php
-  /Users/selwyn/Sites/d9book2/web/index.php
-  /Users/selwyn/Sites/d9book2/web/core/assets/scaffold/files/index.php
-  /Users/selwyn/Sites/d9book2/web/core/tests/Drupal/Tests/Composer/Plugin/Scaffold/fixtures/index.php
-  /Users/selwyn/Sites/d9book2/web/core/tests/fixtures/files/phar-1.phar/index.php



Ignore

Accept

If you accidentally selected the wrong local path, in PhpStorm, go to Settings, PHP, Servers and delete all servers that are displayed. The correct one will be recreated after you retry the operation above.

A screenshot of a code editor window. The file is named 'index.php'. Line 19 is highlighted with a red background and a red arrow points to it from the right, labeled 'Current line'. The code on line 19 is '\$response = \$kernel->handle(\$request);'. The code editor has tabs for 'index.php', 'drush', and 'Drush.php'.

```
1 <?php
2
3 /** @file ...
10
11 use ...
13
14 $autoloader = require_once 'autoload.php'; $autoloader: {registeredLoaders => , vendorDir ...
15
16 $kernel = new DrupalKernel( environment: 'prod' , $autoloader); $autoloader: {registeredLoader ...
17
18 $request = Request::createFromGlobals(); $request: {trustedProxies => , trustedHostPatter ...
19 $response = $kernel->handle($request);
20 $response->send();
21
22 $kernel->terminate($request, $response);
23
```

Once you accept the local path, you should see a highlighted line indicating the current line. The debug window will appear below showing the call stack

A screenshot of a debugger interface. On the left, there's a sidebar with project files like 'release', 'robo', 'security-checker', 'var-dump-server', and a folder 'chi-teck'. Below that is a 'Bookmarks' section with 'd9book2' selected. At the bottom, there's a 'Debug' toolbar with buttons for Console, Debugger, Output, and others. A red arrow points from the 'Debug' button to the 'index.php' tab in the main code editor area. The code editor shows the same PHP code as above, with line 19 highlighted. A red arrow points from the 'Call stack' text to the debug window below. The debug window title is 'index.php:19, {main}()' and contains the text 'Breakpoint reached (a minute ago)'. A red arrow points from the 'Breakpoint reached' message to the status bar at the bottom which says 'Switch frames from anywhere in the IDE with ⌘↑ and ⌘↓'.

release
robo
security-checker
var-dump-server
chi-teck

Bookmarks d9book2 Breakpoints

Debug: index.php

index.php:19, {main}()

Breakpoint reached (a minute ago)

Switch frames from anywhere in the IDE with ⌘↑ and ⌘↓

Git Find Debug TODO Problems Terminal Services

You will also see the variables and watch pane:

The screenshot shows a PhpStorm interface with a code editor at the top containing PHP code. A red arrow points from the code editor down to a 'Variables and watch pane' window below. This pane contains a list of variables with their current values, such as `$_COOKIE`, `$_ENV`, `$_SERVER`, and `$request`. At the bottom of the pane, there is an 'Evaluate expression (e) or add a watch (w)' input field.

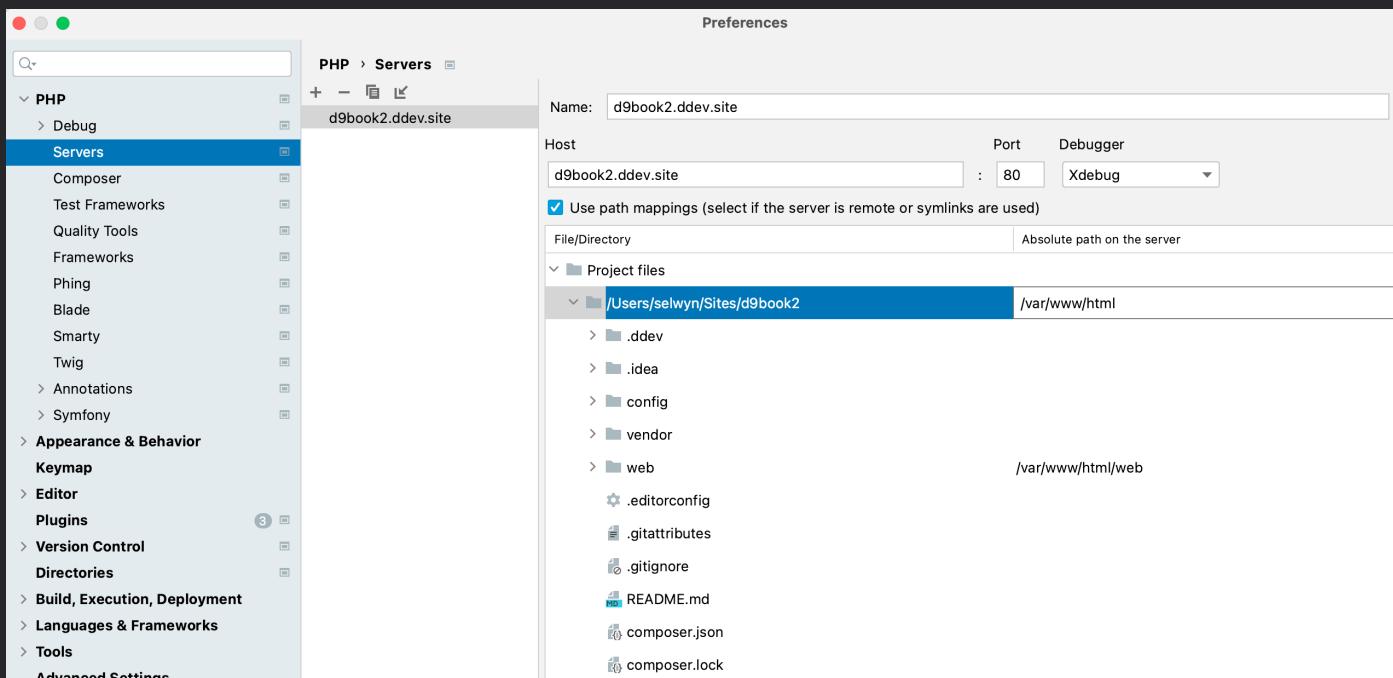
Command line or drush debugging

For command line or drush debugging you must run your code from within the DDEV container. This means you must 'ssh' into the DDEV (Docker) container and execute the command you want to debug this way:

```
$ ddev ssh
```

```
$ vendor/bin/drush status
```

To setup command line debugging, follow the steps above to setup for [Drupal Code debugging](#) to confirm that you have debugging working. Then look in PhpStorm's: settings, PHP, Servers, and select the server you set up from the previous steps. Specify the top level path as shown below. Usually it will be `/var/www/html`



Next open `vendor/drush/drush/src/Drush.php` and specify a breakpoint like this:

The screenshot shows a code editor in PhpStorm with the file `Drush.php` open. A red arrow points from the text "Breakpoint" to a red dot on line 35, which is highlighted in yellow. The code on line 35 is `return self::bootstrapManager()>bootstrap();`. The code editor has a light orange background and displays line numbers from 14 to 46.

```
14     public static function debug(): bool
15     {
16         if (!self::hasService( id: 'output' )) {
17             return false;
18         }
19         return Drush::output()->isDebug();
20     }
21
22     /**
23      * Return the Bootstrap Manager.
24     */
25     public static function bootstrapManager(): BootstrapManager
26     {
27         return self::service( id: 'bootstrap.manager' );
28     }
29
30     /**
31      * Return the Bootstrap object.
32     */
33     public static function bootstrap(): Boot
34     {
35         return self::bootstrapManager()>bootstrap();
36     }
37
38     public static function redispachOptions($input = null)
39     {
40         $input = $input ?: self::input();
41         $command_name = $input->getFirstArgument();
42
43         // $input->getOptions() returns an associative array of option => value
44         $options = $input->getOptions();
45
46         // The 'runtime.options' config contains a list of option names on th c
```

Now in the terminal ssh into the DDEV container and execute drush:

```
$ ddev ssh
```

```
$ vendor/bin/drush status
```

PhpStorm will pop up and display the current line and you can debug to your heart's content:

The screenshot shows the PhpStorm IDE interface. The top navigation bar has tabs for various PHP files like index.php, bin/drush, drush/drush, Drush.php, Robo.php, Container.php, BootstrapManager.php, and DrupalBoot8.php. The main editor window displays the Drush.php file. A red arrow points to a breakpoint icon on line 435. The code is as follows:

```

428 }
429
430 /**
431 * Return the Bootstrap object.
432 */
433 public static function bootstrap(): Boot
434 {
435     return self::bootstrapManager()->bootstrap();
436 }
437
438 public static function redispatchOptions($input = null)
439 {
440     $input = $input ?: self::input();
441     $command_name = $input->getFirstArgument();
442
443     // $input->getOptions() returns an associative array of option => value
444     $options = $input->getOptions();
445
446     // The 'runtime.options' config contains a list of option names on the cli
447     $optionNamesFromCommandLine = self::config()->get( key: 'runtime.options' );
448
449     // Attempt to normalize option names.

```

The bottom status bar shows the current file path: \Drush > Drush > bootstrap(). The bottom right corner shows a tooltip: "Evaluate expression (e) or add a watch (w)".

More at <https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/>

Add a breakpoint in code

To add a breakpoint in code, you can use: xdebug_break()

more at https://xdebug.org/docs/all_functions

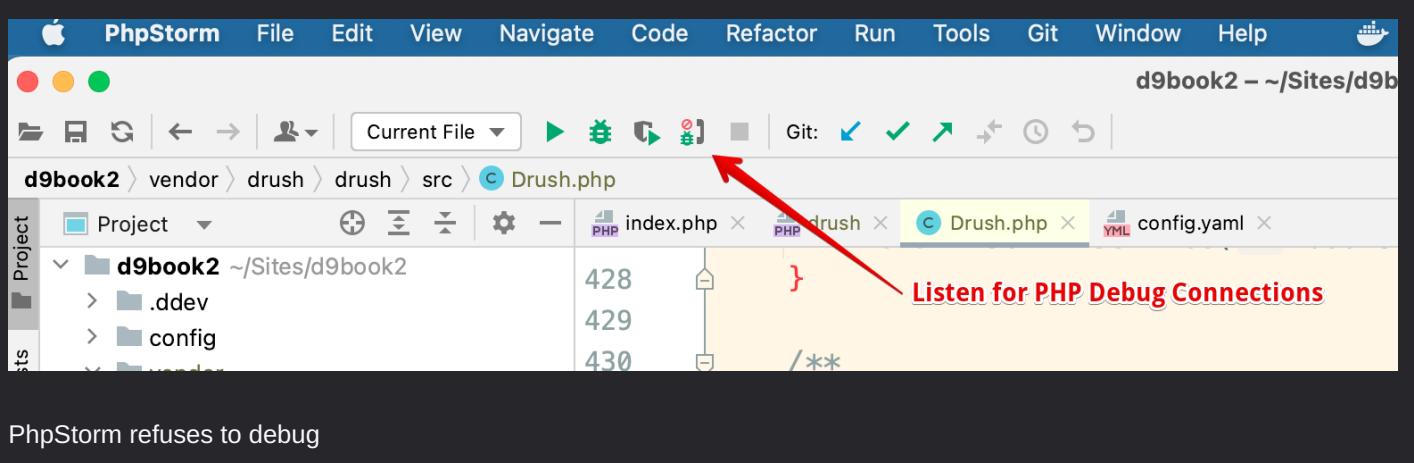
Troubleshooting Xdebug with DDEV

Could not connect to debugging client

When debugging command line e.g. drush commands etc. if you see:

```
$ vendor/bin/drush status
Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9003 (through xdebug.client_host/xdebug.client_port).
```

This means you have not clicked the PhpStorm button: "Start listening for PHP Debug Connections". Just click it and try again



Here are some steps to try:

CURL

Use curl or a browser to create a web request. For example:

```
$ curl https://d9.ddev.site
```

Replace d9.ddev.site with the correct URL for your site.

LOGS

If the IDE doesn't respond, take a look at ddev logs. Use ddev logs to display current logs for the project's web server. See

<https://ddev.readthedocs.io/en/stable/users/basics/commands/#logs> for more.

If you see a message like

```
"PHP message: Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9000 (through xdebug.client_host/xdebug client_port)"
```

then php/xdebug (inside the container) is not able to make a connection to port 9000.

This means you have not clicked the "Start listening for PHP Debug Connections" button in PhpStorm. Just click it and try again.

Note. Port 9003 is more current.

TELNET

With PhpStorm NOT listening for a PHP Debug connection, try to telnet to see what is listening to port 9003 with this:

```
$ telnet d9book2.ddev.site 9003
```

You should get connection refused. If you get a connection (see below) something is listening on port 9003 and you should either disable it or use a different port. Note. You must set that in both PhpStorm as well as DDEV.

This shows a connection succeeding. You can try it with PhpStorm listening for a debug connection:

```
$ telnet d9book2.ddev.site 9003
```

```
Trying 127.0.0.1...
```

```
Connected to d9book2.ddev.site.
```

```
Escape character is '\n'.
```

Use Control] to exit and then type quit to return to exit telnet.

On a mac, use sudo lsof -i :9003 -sTCP:LISTEN to find out what is listening on that port and stop it, or change the xdebug port and configure both DDEV and PhpStorm to use the new one .

More about changing ports at <https://ddev.readthedocs.io/en/stable/users/debugging-profiling/step-debugging/#using-xdebug-on-a-port-other-than-the-default-9003>

Note. In the past, php-fpm was likely to be one of the apps using port 9000.

IS XDEBUG ENABLED?

- To check to make sure that Xdebug is enabled, you can use `php -i | grep xdebug` inside the container. You can also use other techniques to view the output of `phpinfo()`, including Drupal's `admin/reports/status/php`. Below you can see the expected output when Xdebug is enabled assuming you have Xdebug v3.2.0 or later.

```
$ php -i | grep "xdebug.remote_enable"
```

```
xdebug.remote_enable => (setting renamed in Xdebug 3) => (setting renamed in Xdebug 3)
```

See <https://ddev.readthedocs.io/en/stable/users/step-debugging>

What is listening on the debug port?

To check if something is listening on port 9003, it's best to use `lsof` as it will actually list the name of the process listening.

i.e. Here we see phpstorm listening:

```
lsof -i TCP:9003
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
phpstorm 78897 selwyn 209u IPv4 0x4d118c80f54422d7 0t0 TCP *:9003 (LISTEN)
```

You can also use `nc` and `netstat` but they are not quite as informative:

```
$ nc -z localhost 9003
Connection to localhost port 9003 [tcp/*] succeeded!
```

The message `Connection to localhost port 9003 [tcp/*] succeeded!` means that there is a program listening on port 9003. If you get nothing when you type the command, then nothing is listening.

Here `netstat` reports that something is listening on port 9003. Again, if you get nothing when you type the command, then nothing is listening.

```
$ netstat -an | grep 9003
tcp4      0      0  *:9003          *.*        LISTEN
```

Here is an example running `lsof` and finding php-fpm listening on port 9000

```
$ lsof -i TCP:9000
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
php-fpm 732 selwyn 7u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)

php-fpm 764 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)

php-fpm 765 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
localhost:cslistener (LISTEN)
```

Enable twig debugging output in source

In `sites/default/development.services.yml` in the parameters, `twig.config`, set `debug: true`. See `core.services.yml` for lots of other items to change for development.

```
# Local development services.
#
parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    debug: true
    auto_reload: true
    cache: false

# To disable caching, you need this and a few other items
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in `settings.local.php`:

```
/**  
 * Enable local development services.  
 */  
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in `settings.local.php` with:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';
```

Devel and Devel Kint Extras

From <https://www.webwash.net/how-to-print-variables-using-devel-and-kint-in-drupal/>

Setup

We need both the Devel and Devel Kint Extras module. [Devel Kint Extras](#) ships with the kint-php library so installing this via Composer will take care of it automatically.

Install using Composer:

```
composer require drupal/devel drupal/devel_kint_extras
```

Enable both with the following Drush command:

```
$ ddev drush en devel_kint_extras -y
```

Finally, enable Kint Extended as the Variables Dumper. To do this go to:

admin/config/development/devel

and select Kint Extender and Save the configuration

Add kint to a custom module

```
function custom_kint_preprocess_page(&$variables) {  
  kint($variables['page']);  
}
```

Dump variables in a TWIG template

```
{{ kint(attributes) }}
```

Kint::dump

From [Migrate Devel contrib module](#), in `docroot/modules/contrib/migrate-devel/src/EventSubscriber/MigrationEventSubscriber.php`

This is used in migrate to dump the source and destination values.

```
// We use kint directly here since we want to support variable naming.  
kint_require();  
\Kint::dump($Source, $Destination, $DestinationIDValues);
```

Set max levels to avoid running out of memory

Kint can run really slowly so you may have to set `maxLevels` this way:

Add this to `settings.local.php`

```
// Change kint maxLevels setting:  
include_once(DRUPAL_ROOT . '/modules/contrib/devel/kint/kint.class.php');  
if(class_exists('Kint')){  
    // Set the maxlevels to prevent out-of-memory. Currently there doesn't seem to be a cleaner way to set this:  
    Kint::$maxLevels = 4;  
}
```

Resources

- [DDEV Documentation](#)
 - [Debugging with Xdebug in DDEV docs](#)
 - [Debug Drush commands with PhpStorm at](#)
 - [Configuring PhpStorm Drupal.org docs updated September 2022](#)
 - [Debugging Drush commands](#)
 - [DDEV docs on using a different port for debugging](#)
 - [How to setup Devel and Kint on Drupal 9 by Alex Aug 2021](#)
-
-

[Back to top](#)

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

Page last modified: May 14 2023.

[Edit this page on GitHub](#)

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

Development

TABLE OF CONTENTS

- [Local Drupal site setup](#)
 - [First Option](#)
 - [Second Option](#)
- [Checking Your Permissions](#)
- [Converting existing site \(non-composer based\) to use composer](#)
- [Composer best practices for Drupal 8](#)
- [DDEV](#)
 - [Local config - your .ddev/config.local.yaml](#)
 - [NFS](#)
 - [Mutagen](#)
 - [setup aliases in ddev](#)
 - [Upgrading ddev](#)
 - [Show others your ddev local site](#)
 - [Email Capture and Review](#)
 - [DDEV and Xdebug](#)
 - [Command line or drush debugging](#)
 - [Use drush commands in your shell with DDEV](#)
 - [Load your data from an Acquia site](#)
 - [Cleanup some disk space](#)
 - [Accessing specific containers](#)
- [DDEV Troubleshooting](#)
 - [Running out of docker disk space](#)
 - [DDEV won't start](#)
- [PHPStorm](#)
 - [Setting up PHPStorm and Drupal](#)
 - [PHPStorm and Xdebug](#)
 - [add a breakpoint in code](#)
 - [Collecting PhpStorm debugging logs](#)
- [Troubleshooting Xdebug with DDEV](#)
- [What is listening on port 9000?](#)
- [Setup settings.local.php and disable Cache](#)
- [Development.services.yml](#)
- [Enable twig debugging output in source](#)
- [Kint](#)
 - [Setup](#)
 - [Add kint to a custom module](#)
 - [Dump variables in a TWIG template](#)
 - [Kint::dump](#)
 - [Set max levels to avoid running out of memory](#)
- [Replacing deprecated functions](#)
- [Missing module](#)
- [You have requested a non-existent service](#)
- [Resources](#)

Local Drupal site setup

Local development is best done using containers and [DDEV](#). Setting up a local site is a completely painless process.

Pick one of these options after installing Docker and Ddev:

FIRST OPTION

Using the DDEV Quickstart guides to install Drupal, Wordpress, TYPO3, Backdrop, Magento, Laravel etc. at <https://ddev.readthedocs.io/en/stable/users/quickstart/#drupal>

```
mkdir my-drupal10-site
cd my-drupal10-site
ddev config --project-type=drupal10 --docroot=web --create-docroot
ddev start
ddev composer create "drupal/recommended-project" --no-install
ddev composer require drush/drush --no-install
ddev composer install
ddev drush site:install -y
ddev drush uli
ddev launch
```

OR

SECOND OPTION

From https://www.drupal.org/docs/official_docs/en/_local_development_guide.html

Start by specifying your SITE_NAME using export:

```
export SITE_NAME=d9site
export SITE_NAME=clientsite
```

Here are all the steps:

```
export SITE_NAME=my-drupal-site
composer create-project drupal/recommended-project $SITE_NAME
cd $SITE_NAME
ddev config --docroot=web --project-name=$SITE_NAME --project-type=drupal9
ddev start
ddev exec drush site-install --account-name=admin --account-pass=admin
```

Checking Your Permissions

During the wizard installation, or when your welcome page first loads, you might see a warning about the permissions settings on your /sites/web/default directory and one file inside that directory: settings.php.

After the installation script runs, [Drupal will try to set the web/sites/default directory permissions to read and execute for all groups](#): this is a 555 permissions setting. It will also attempt to set permissions for default/settings.php to read-only, or 444. If you encounter this warning, run these two chmod commands from your project's root directory. Failure to do so poses a security risk:

```
chmod 555 web/sites/default
```

```
chmod 444 web/sites/default/settings.php
```

To verify that you have the correct permissions, run this ls command with the a, l, h, and d switches and check that your permissions match the following output:

```
$ ls -alhd web/sites/default web/sites/default/settings.php  
  
dr-xr-xr-x 8 sammy staff 256 Jul 21 12:56 web/sites/default  
-r--r--r-- 1 sammy staff 249 Jul 21 12:12 web/sites/default/settings.php
```

You are now ready to develop a Drupal website on your local machine.

Converting existing site (non-composer based) to use composer

[Composerize Drupal](#)

Also for [manual steps](#)

Composer best practices for Drupal 8

<https://www.lullabot.com/articles/drupal-8-composer-best-practices>

DDEV

For local Docker container development on any platform, there is no better tool than DDEV. This is a [well-documented, well-supported](#) tool by the Amazing Randy Fay. You can get help from him or some of the other friendly folks on [Discord](#) almost instantly.

From the docs:

- Lots of built-in help: ddev help and ddev help <command>. You'll find examples and explanations.
- [DDEV Documentation](#)
- [DDEV Stack Overflow](#) for support and frequently asked questions. We respond quite quickly here and the results provide quite a library of user-curated solutions.
- [DDEV issue queue](#) for bugs and feature requests
- Interactive community support on [Discord](#) for everybody, plus sub-channels for CMS-specific questions and answers.
- [ddev-contrib](#) repo provides a number of vetted user-contributed recipes for extending and using DDEV. Your contributions are welcome.
- [awesome-ddev](#) repo has loads of external resources, blog posts, recipes, screencasts, and the like. Your contributions are welcome.
- [Twitter with tag #ddev](#) will get to us, but it's not as good for interactive support, but we'll answer anywhere.

Local config - your .ddev/config.local.yaml

From https://ddev.readthedocs.io/en/stable/users/extend/config_yaml

- You can override the config.yaml with extra files named config.*.yaml. For example, many teams use config.local.yaml for configuration that is specific to one environment, and that is not intended to be checked into the team's default config.yaml.
- You could add a config.selwyn.yaml for Selwyn-specific values.
- Use ddev start (or ddev restart) after making changes to get the changes to take effect.

In the endless quest for speed in local development, try using NFS or Mutagen on MAC OS. Apparently the WSL2 setup on Windows 10/11 is the fastest performer for DDEV at th time of this writing.

NFS

```
router_http_port: \"80\"\nrouter_https_port: \"443\"\ntimezone: America/Chicago\n# and for nfs\nnfs_mount_enabled: true
```

MUTAGEN

```
# instead of nfs, use mutagen
nfs_mount_enabled: false
mutagen_enabled: true
```

setup aliases in ddev

I love short linux aliases like `ll` (or just `l`) for listing files. If you spend time poking around the file system in your containers this makes life so much better. A cool new feature since Ddev v15.1 lets you add aliases using this technique

Use `ddev ssh` to “`ssh`” into the container and then type `ll` to list the files in a directory.

Either copy `.ddev/homeadditions/bash_aliases.example` to `.ddev/homeadditions/bash_aliases` and add them there!

OR

Create a file `.ddev/homeadditions/.bash_aliases` with these contents: note. those are the letter L lower case (as in lima).

```
alias ll="ls -lhAp"
alias l="ls -lhAp"
```

Note. don't use `.homeadditions` - use the `homeadditions`.

Upgrading ddev

After you install a new version of ddev, run `ddev stop` and then `ddev config` to reconfigure things for your project. Just press enter for all the questions. It keeps things rolling smoothly. Run `ddev start` to start it all back up again

Show others your ddev local site

Sharing your DDEV-Local site via a public URL using `ddev share` and `ngrok`. by Mike Anello

[https://www.drupaleeasy.com/blogs/ultimike/2019/06/sharing-your-ddev-local-site-public-url-using-ddev-share-and-ngrok](https://www.drupaleasy.com/blogs/ultimike/2019/06/sharing-your-ddev-local-site-public-url-using-ddev-share-and-ngrok)

Email Capture and Review

MailHog is a mail catcher which is configured to capture and display emails sent by PHP in the development environment.

After your project is started, access the MailHog web interface at its default port:

`http://mysite.ddev.site:8025`

Please note this will not intercept emails if your application is configured to use SMTP or a 3rd-party ESP integration. If you are using SMTP for outgoing mail handling (Swiftmailer or SMTP modules for example), update your application configuration to use localhost on port 1025 as the SMTP server locally in order to use MailHog.

`ddev launch -m` will launch the MailHog UI.

DDEV and Xdebug

This is a magical match made in heaven. To enable or disable Xdebug use

```
$ ddev xdebug on
```

and

```
$ ddev xdebug off
```

Note. This will slow everything down because xdebug has a significant performance impact so be sure to disable it when you are finished with your debugging session.

In phpstorm, you can uncheck the following settings:

- force break at first line when no path mapping is specified
- force break at first line when a script is outside the project

Note. we usually use port 9000 for xdebug look in `.ddev/php/xdebug_report_port.ini` for the real port settings. Recently for a project I found it set to 11011

The contents of the file are:

```
[PHP]  
  
xdebug.remote_port=11011
```

For phpstorm, if you start listening for a debug connection, it should automatically try to create a debug server config for you. If it doesn't manually create one

e.g name: tea.ddev.site

host tea.ddev.site

port: 80

debugger: xdebug

check use path mappings

for docroot specify: /var/www/html/docroot (i.e. wherever index.php is)

Command line or drush debugging

For command line or drush debugging (xdebug, phpstorm)

```
ddev ssh  
  
export PHP_IDE_CONFIG="serverName=d8git.ddev.site"
```

or

```
export PHP_IDE_CONFIG="serverName=inside-mathematics.ddev.site"  
  
confirm debug is turned on
```

```
php -i | grep debug
```

You should see:

```
xdebug support => enabled
```

Also you can confirm the port

set a server in phpstorm that matches the name d8git.ddev.site OR inside-mathematics.ddev.site.

Configure the server to use path mappings

/Users/selwyn/Sites/ddev 82 ---> /var/www/html

click listen for debug connections button

set breakpoint and run

replace d8git.ddev.site with the name of your project

NOTE!!!!. You must execute drush from the vendor dir or you will always be ignored:

```
./vendor/drush/drush/drush fixmat
```

If it doesn't seem to work, try enable Break at first line in PHP scripts - something will always stop then.

more at <https://stackoverflow.com/questions/50283253/how-can-i-step-debug-a-drush-command-with-ddev-and-phpstorm>

Use drush commands in your shell with DDEV

If you do local development, you can use syntax like `ddev drush cst` to execute drush commands in the container. This is slower than running on your native system because they are executed in the container. I prefer using drush directly on the host computer.

To do this install PHP as well drush launcher. Once these are working, you can `cd` into the project directory and issue commands like `drush cr`, `drush cst` OR `drush cim -y etc`. It is so very quick and smooth. (Note. this is the case with MacOS and Linux but I don't really know how it works on Windows.)

[Details for Drush Launcher](#)

From [Installation of Drush Launcher](#)

- To be able to call drush from anywhere, install the [Drush Launcher](#). Launcher is a small program which listens on your \$PATH and hands control to a site-local Drush that is in the /vendor directory of your Composer project.

Luckily, DDEV enables this functionality by default (Thanks Randy!)

Load your data from an Acquia site

Using the [drush aliases](#) assuming the site is called `abc` and you want the `prod` (production) database:

```
$ drush @abc.prod sql-dump >dbprod.sql  
$ gzip dbprod.sql  
$ ddev import-db --src=dbprod.sql.gz
```

Of course this works with any site where you've set up your [drush aliases](#).

Cleanup some disk space

Free up disk space used by previous docker image versions. This does no harm.

```
ddev delete images
```

also

```
docker system prune
```

and

```
docker image prune -a
```

List all docker volumes

```
docker volume ls
```

[DDEV General cleanup topic](#)

Accessing specific containers

To ssh into a specific service e.g. from a `docker-composer.chromedriver.yml` the service is listed under "services:" like:

```
services:  
  chromedriver
```

Use

```
ddev ssh -s chromedriver
```

or for selenium, use:

```
ddev ssh -s selenium
```

DDEV Troubleshooting

Running out of docker disk space

if ddev won't start and shows:

```
Creating ddev-router ... done
Failed to start ddev82: db container failed: log=, err=container exited, please use 'ddev logs -s db' to find out why it failed
```

Looking in the log, you might see:

```
preallocating 12582912 bytes for file ./ibtmp1 failed with error 28
2020-03-16 14:27:54 140144158233920 [ERROR] InnoDB: Could not set the file size of './ibtmp1'. Probably out of disk space
```

That is the clue.

You can kill off images using

```
ddev delete images
```

or the more drastic

```
docker rmi -f $(docker images -q)
```

Q. Deleting the images: Does that mean it will delete the db snapshots? A. No, docker images are the versioned images that come from dockerhub, they're always replaceable.

Absolutely nothing you do with ddev will delete your snapshots - you have to remove them manually

They're stored in .ddev/db_snapshots on the host (under each project)

also

```
docker system prune
```

and

```
docker system prune --volumes
```

prunes every single thing, destroys all ddev databases and your composer cache.

DDEV won't start

ddev pull or ddev start failed with error something like:

```
Pull failed: db container failed: log=, err=health check timed out: labels map[com.ddev.site-name:inside-mathematics com.docker.compose.service:db] timed out without becoming healthy
```

Or like this:

```
$ ddev start
Starting inside-mathematics...
Pushing mkcert rootca certs to ddev-global-cache
Pushed mkcert rootca certs to ddev-global-cache
Creating ddev-inside-mathematics-db ... done
Creating ddev-inside-mathematics-dba ... done
Creating ddev-inside-mathematics-web ... done

Creating ddev-router ... done
```

```
Failed to start inside-mathematics: db container failed: log=, err=health check timed out: labels map[com.ddev.site-name:inside-mathematics com.docker.compose.service:db] timed out without becoming healthy
```

This is almost always caused by a corrupted database, most often in a larger database. Since v0.17.0, this is generally only caused by docker being shut down in an ungraceful way. Unfortunately, both Docker for Windows and Docker for Mac shut down without notifying the container during upgrade, with a manual Docker exit, or at system shutdown. It can be avoided by stopping or removing your projects before letting Docker exit.

To fix, ddev remove --remove-data, then ddev start. This may fail and suggest this bazooka version:

```
ddev stop --remove-data --omit-snapshot
```

PHPStorm

All the PHPStorm Drupal magic is at https://www.jetbrains.com/help/phpstorm/drupal-support.html#view_drupal_api_documentation

Setting up PHPStorm and Drupal

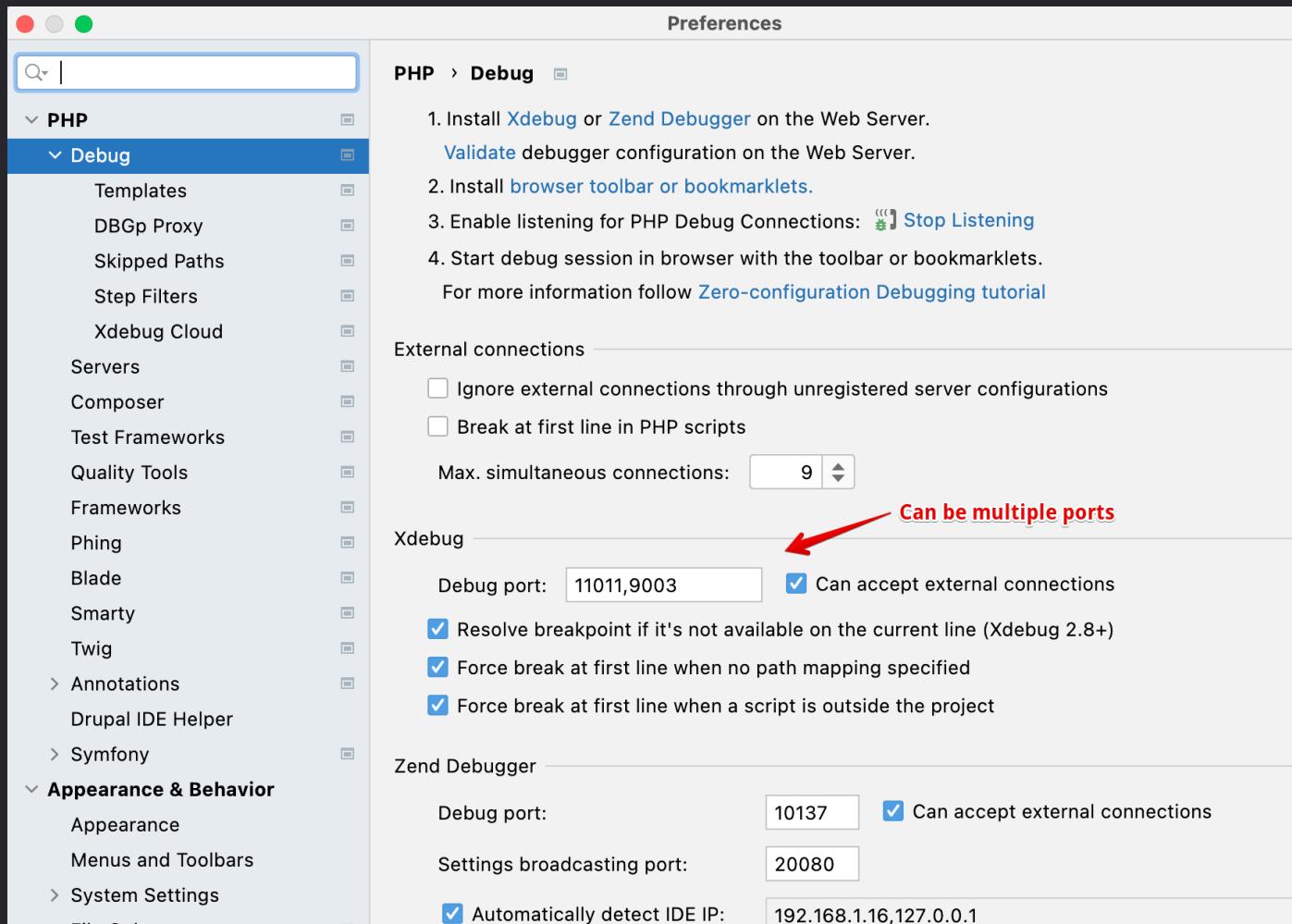
<https://www.drupal.org/docs/develop/development-tools/configuring-phpstorm>

PHPStorm and Xdebug

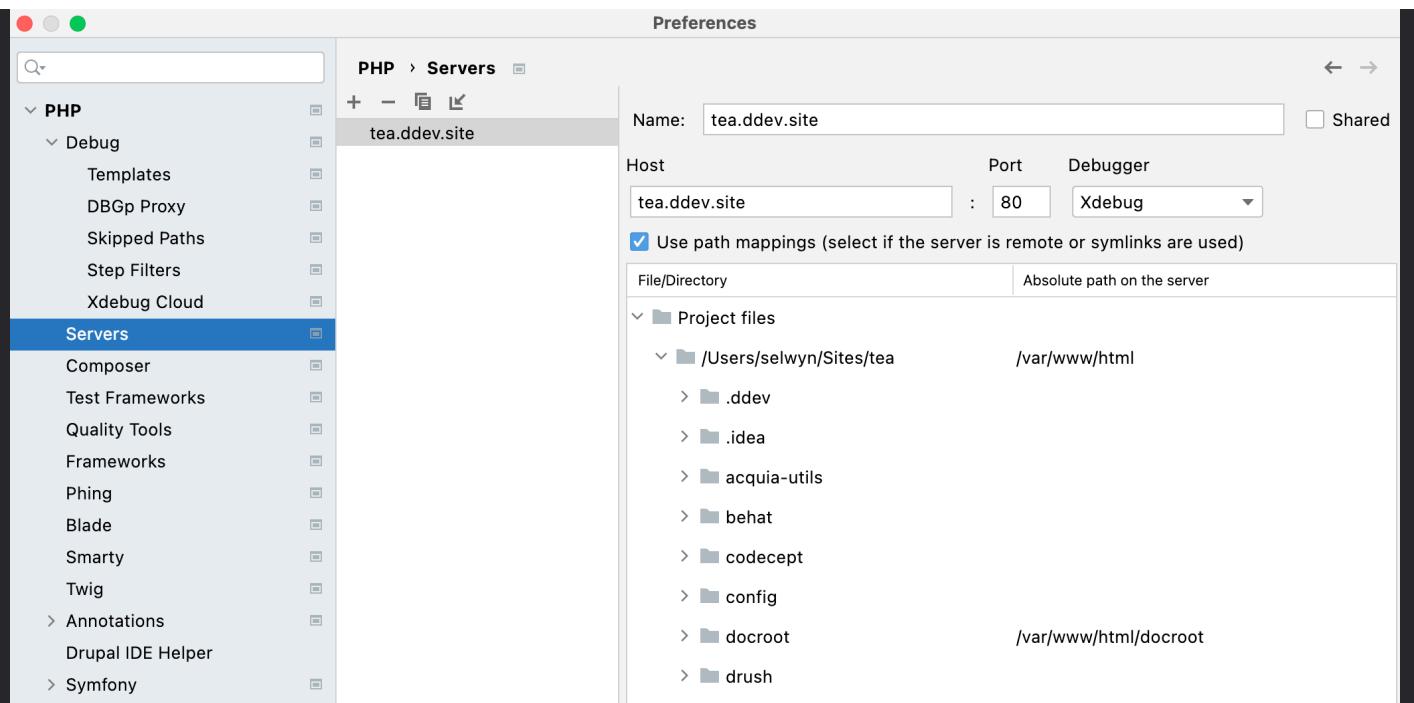
Debugging drush commands at <https://www.jetbrains.com/help/phpstorm/drupal-support.html#debugging-drush-commands>

PHPStorm has a series of instructions for [configuring PHPStorm with Xdebug](#) but unfortunately, nothing specifically on using it with DDEV. Fortunately it doesn't require any special setup for it to work.

Some settings I use

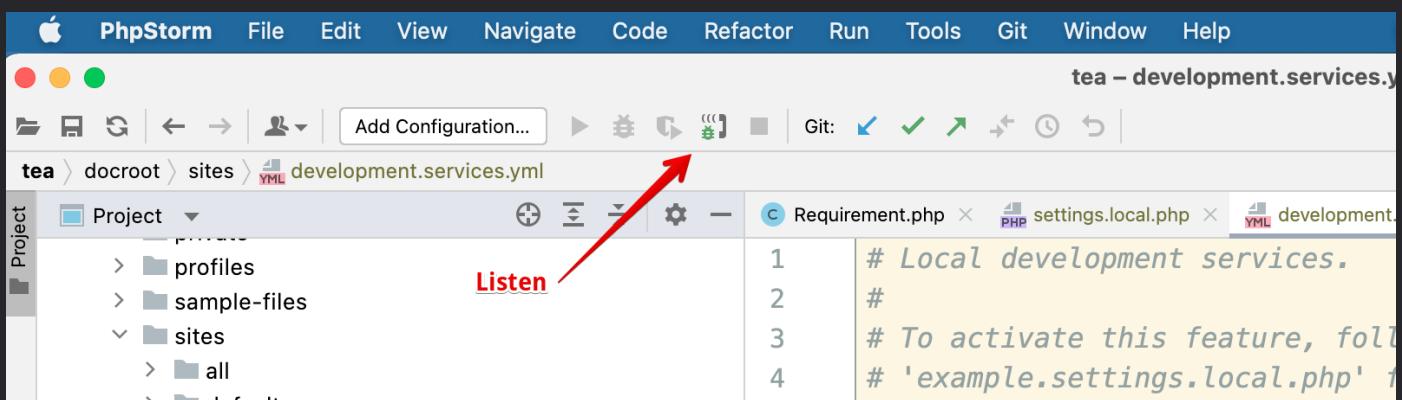


And for this project



If phpstorm doesn't stop when you set a breakpoint on some code, try deleting the server from the config debug, php, servers.

Make sure PHPStorm is listening by clicking the listen button



When you try again it will be recreated but you will probably need to specify the path (from the image above).

ADD A BREAKPOINT IN CODE

To add a breakpoint in code, use

```
xdebug_break()
```

more at https://xdebug.org/docs/all_functions

Collecting PhpStorm debugging logs

- In the Settings/Preferences dialog (⌘,), go to PHP.
- From the PHP executable list, choose the relevant PHP interpreter and click next to it. In the CLI Interpreters dialog that opens, click the Open in Editor link next to the Configuration file: <path to php.ini> file. Close all the dialogs and switch to the tab where the php.ini file is opened.
- In the php.ini, enable Xdebug logging by adding the following line:
- For Xdebug 3xdebug.log="path_to_log/xdebug.log" The log file contains the raw communication between PhpStorm and Xdebug as well as any warnings or errors:
- <https://www.jetbrains.com/help/phpstorm/troubleshooting-php-debugging.html#collecting-logs>

Troubleshooting Xdebug with DDEV

- Use curl or a browser to create a web request. For example, curl <https://d9.ddev.site>
- If the IDE doesn't respond, take a look at ddev logs (`ddev logs`). If you see a message like ""PHP message: Xdebug: [Step Debug] Could not connect to debugging client. Tried: host.docker.internal:9000 (through xdebug.client_host/xdebug.client_port)" then php/xdebug (inside the container) is not able to make a connection to port 9000.
- In PhpStorm, disable the "listen for connections" button so it won't listen. Or just exit PhpStorm. With another IDE like vscode, stop the debugger from listening.
- ddev ssh: Can telnet host.docker.internal 9000 connect? If it does, you have something else running on port 9000, probably php-fpm. On the host, use `sudo lsof -i :9000 -sTCP:LISTEN` to find out what is there and stop it, or change the xdebug port and configure PhpStorm to use the new one . Don't continue debugging until your telnet command does not connect.
- Check to make sure that Xdebug is enabled. You can use `php -i | grep Xdebug` inside the container, or use any other technique you want that gives the output of `phpinfo()`, including Drupal's `admin/reports/status/php`. You should see with Xdebug v2.9.6, Copyright (c) 2002-2020 and `php -i | grep "xdebug.remote_enable"` should give you `xdebug.remote_enable: On`.

<https://ddev.readthedocs.io/en/stable/users/step-debugging/>

What is listening on port 9000?

To check if something is listening on port 9000 (the default port for xdebug) it's best to use

```
$ lsof -i TCP:9000
```

it will actually list the name of the process listening

i.e.

```
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
phpstorm 13361 selwyn 81u IPv6 0x5d4d30caf0be07d 0t0 TCP *:cslistener (LISTEN)
```

Another option is

```
nc -z localhost 9000
```

If it says:

Connection to localhost port 9000 [tcp/cslistener] succeeded!

this means something is listening. If you get nothing, then nothing is listening.

You can also run network utility, scan port 9000 to 9003 on 127.0.0.1 (localhost)

What could be listening on port 9000?

```
$ netstat -an | grep 9000
```

```
tcp4 0 0 127.0.0.1.9000 *.* LISTEN
```

Other options include:

```
$ lsof -i TCP:9000
```

Which reports that php-fpm is listening.

```
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
```

```
php-fpm 732 selwyn 7u IPv4 0x4120ed57a07e871f 0t0 TCP
```

```
localhost:cslistener (LISTEN)
```

```
php-fpm 764 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
```

```
localhost:cslistener (LISTEN)
```

```
php-fpm 765 selwyn 8u IPv4 0x4120ed57a07e871f 0t0 TCP
```

```
localhost:cslistener (LISTEN)
```

Setup settings.local.php and disable Cache

From [Disabling cache during development](#)

1. Copy, rename, and move the sites/example.settings.local.php to sites/default/settings.local.php:

```
$ cp sites/example.settings.local.php sites/default/settings.local.php
```

2. Open sites/default/settings.php and uncomment these lines:

```
if(file_exists($app_root . '/' . $site_path . '/settings.local.php')) {  
  include $app_root . '/' . $site_path . '/settings.local.php';  
}
```

This will include the local settings file as part of Drupal's settings file.

3. Open settings.local.php and make sure development.services.yml is enabled.

```
$settings['container_yamls'][0] = DRUPAL_ROOT . '/sites/development.services.yml';
```

By default development.services.yml contains the settings to disable Drupal caching:

```
services:  
  cache.backend.null:  
    class: Drupal\Core\Cache\NullBackendFactory
```

NOTE: Do not create development.services.yml, it exists under /sites

4. In settings.local.php change the following to be TRUE if you want to work with enabled css- and js-aggregation:

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

5. Uncomment these lines in settings.local.php to disable the render cache and disable dynamic page cache:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';  
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
```

Add the following lines to your sites/default/settings.local.php

```
$settings['cache']['bins']['page'] = 'cache.backend.null';
```

If you do not want to install test modules and themes, set the following to FALSE:

```
$settings['extension_discovery_scan_tests'] = FALSE;
```

6. Open sites/development.services.yml in the sites folder and add the following block to disable the twig cache and enable twig debugging:

```
parameters:  
  twig.config:  
    debug: true  
    auto_reload: true  
    cache: false
```

NOTE: If the parameters section is already present in the development.services.yml file, append the twig.config section to it.

7. Rebuild the Drupal cache (`drush cr`) otherwise your website will encounter an unexpected error on page reload.

Development.services.yml

I usually develop with this in `sites/default/development.services.yml`

```

# Local development services.
#
# To activate this feature, follow the instructions at the top of the
# 'example.settings.local.php' file, which sits next to this file.

parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    # Twig debugging:
    #
    # When debugging is enabled:
    # - The markup of each Twig template is surrounded by HTML comments that
    #   contain theming information, such as template file name suggestions.
    # - Note that this debugging markup will cause automated tests that directly
    #   check rendered HTML to fail. When running automated tests, 'debug'
    #   should be set to FALSE.
    # - The dump() function can be used in Twig templates to output information
    #   about template variables.
    # - Twig templates are automatically recompiled whenever the source code
    #   changes (see auto_reload below).
    #
    # For more information about debugging Twig templates, see
    # https://www.drupal.org/node/1906392.
    #
    # Not recommended in production environments
    # @default false
    debug: true
    # Twig auto-reload:
    #
    # Automatically recompile Twig templates whenever the source code changes.
    # If you don't provide a value for auto_reload, it will be determined
    # based on the value of debug.
    #
    # Not recommended in production environments
    # @default null
    #   auto_reload: null
    auto_reload: true
    # Twig cache:
    #
    # By default, Twig templates will be compiled and stored in the filesystem
    # to increase performance. Disabling the Twig cache will recompile the
    # templates from source each time they are used. In most cases the
    # auto_reload setting above should be enabled rather than disabling the
    # Twig cache.
    #
    # Not recommended in production environments
    # @default true
    cache: false
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory

```

Make sure the following is in docroot/sites/default/settings.local.php

```

/**
 * Enable local development services.
 */
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';

```

Enable twig debugging output in source

In sites/default/development.services.yml Set twig.config debug:true. See core.services.yml for lots of other items to change for development

```
# Local development services.  
#  
parameters:  
  http.response.debug_cacheability_headers: true  
  twig.config:  
    debug: true  
    auto_reload: true  
    cache: false  
  
  # To disable caching, you need this and a few other items  
  services:  
    cache.backend.null:  
      class: Drupal\Core\Cache\NullBackendFactory
```

to enable put the following in settings.local.php:

```
/**  
 * Enable local development services.  
 */  
$settings['container_yamls'][0] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in settings.local.php with:

```
$settings['cache']['bins']['render'] = 'cache.backend.null';
```

Kint

From <https://www.webwash.net/how-to-print-variables-using-devel-and-kint-in-drupal/>

Setup

We need both the Devel and Devel Kint Extras module. [Devel Kint Extras](#) ships with the kint-php library so installing this via Composer will take care of it automatically.

Install using Composer:

```
$ composer require drupal/devel drupal/devel_kint_extras
```

Enable both with the following Drush command:

```
$ drush en devel_kint_extras -y
```

Finally, enable Kint Extended as the Variables Dumper. To do this go to admin/config/development/devel and select Kint Extender and Save the configuration.

Add kint to a custom module

```
function custom_kint_preprocess_page(&$variables) {  
  kint($variables['page']);  
}
```

Dump variables in a TWIG template

```
{{ kint(attributes) }}
```

Kint::dump

From [Migrate Devel contrib module](#), in /docroot/modules/contrib/migrate_devel/src/EventSubscriber/MigrationEventSubscriber.php

This is used in migrate to dump the source and destination values.

```
// We use kint directly here since we want to support variable naming.  
kint_require();  
\Kint::dump($Source, $Destination, $DestinationIDValues);
```

Set max levels to avoid running out of memory

Kint can run really slowly so you may have to set maxLevels this way:

Add this to settings.local.php

```
// Change kint maxLevels setting:  
include_once(DRUPAL_ROOT . '/modules/contrib/devel/kint/kint/Kint.class.php');  
if(class_exists('Kint')){  
  // Set the maxlevels to prevent out-of-memory. Currently there doesn't seem to be a cleaner way to set this:  
  Kint::$maxLevels = 4;  
}
```

Replacing deprecated functions

<https://drupal.stackexchange.com/questions/144147/get-taxonomy-terms>

Missing module

from: <https://www.drupal.org/node/2487215>

If you see a PHP warning such as The following module is missing from the file system... (or similar) on your site, You can remove it with:

```
$ drush sql-query "DELETE FROM key_value WHERE name='module_name';"
```

You have requested a non-existent service

```
Symfony\Component\DependencyInjection\Exception\ServiceNotFoundException: You have requested a non-existent service "lingotek.content_translation". in /var/www/vendor/
```

Sometimes, when drush cr throws errors like that try drush sqlc and then truncate cache_bootstrap and truncate cache_discovery.

Resources

- [Composer best practices for Drupal 8 from Lullabot Jan 2018](#)
- [Why DDEV by Randy Fay \(Author of DDEV\) from Dec 2022](#)
- [How to setup Devel and Kint on Drupal 9 by Alex Aug 2021](#)

[Back to top](#)

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

Page last modified: May 14 2023.

[Edit this page on GitHub](#)

Drush

TABLE OF CONTENTS

- [Overview](#)
- [Drush commands](#)
 - [Generate commands](#)
 - [Drush command Example](#)
 - [Drush commands and parameters](#)
 - [Output messages on screen](#)
- [Drush Scripts](#)
 - [Run your scripts](#)
 - [Arguments example](#)
 - [Arguments example2](#)
 - [Hello world example](#)
 - [Database query example](#)
 - [entityQuery example](#)
- [Drush does that?](#)
 - [List blocks](#)
 - [Generate code](#)
 - [Sanitize Databases](#)
 - [Delete nodes](#)
 - [Delete redirects](#)
 - [Watch the watchdog log](#)
 - [Read Drupal config](#)
 - [Check if current config matches exported config](#)
 - [User Login](#)
 - [Drupal:directory](#)
- [Global Install](#)
- [Drush aliases](#)
 - [Example Drush alias file](#)
 - [Example Acquia Drush alias files](#)
 - [Example Drush alias file from Drush project](#)
- [Drush rsync](#)
- [Drush SQL-sync](#)
 - [SQL Queries](#)
- [Drush launcher](#)
- [Drupal 7 Drush scripts](#)
- [Resources](#)

views 177

Overview

Drush is a command line shell and Unix scripting interface for Drupal. Your life will go much better if you use Drush! Drush core ships with lots of [useful commands](#) and [generators](#). Similarly, it runs update.php, executes SQL queries, runs content migrations, and miscellaneous utilities like cron or cache rebuild. Drush can be extended by [3rd party commandfiles](#).

From <https://www.drush.org/latest/>

Drush is extensible with the ability to create drush commands and drush scripts.

Drush commands

These are new commands that you can add to your modules to allow drush to do useful things. Modules sometimes include drush commands e.g. search api (https://www.drupal.org/project/search_api) More about this below.

Modules that have drush 9 commands need the following

- 1 drush.services.yml
- 2 The implementation class which extends the DrushCommands class in module/src/Commands.
- 3 In the class, you annotate the function with the command
- 4 A section in the composer.json file referencing the extra, drush, services.

See below for examples of each.

Generate commands

Drush can generate all the files to create a new command. Use:

```
$ drush generate drush-command-file
```

Or drush dcf

Drush command Example

The (Search API module)[https://www.drupal.org/project/search_api] has a set of Drush commands.

Here is web/modules/contrib/search_api/drush.services.yml

```
services:  
  search_api.commands:  
    class: Drupal\search_api\Commands\SearchApiCommands  
    arguments: ['@entity_type.manager', '@module_handler', '@event_dispatcher']  
    tags:  
      - { name: drush.command }
```

Here is the beginning of the web/modules/contrib/search_api/src/Commands/SearchApiCommands.php which includes the class SearchApiCommands which extends the DrushCommands class:

```

<?php

namespace Drupal\search_api\Commands;

use Consolidation\OutputFormatters\StructuredData\RowsOfFields;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Drupal\Core\Extension\ModuleHandlerInterface;
use Drupal\search_api\Contrib\RowsOfMultiValueFields;
use Drupal\search_api\Utility\CommandHelper;
use Drush\Commands\DrushCommands;
use Psr\Log\LoggerInterface;
use Symfony\Contracts\EventDispatcher\EventDispatcherInterface;

/**
 * Defines Drush commands for the Search API.
 */
class SearchApiCommands extends DrushCommands {

  /**
   * The command helper.
   *
   * @var \Drupal\search_api\Utility\CommandHelper
   */
  protected $commandHelper;

  /**
   * Constructs a SearchApiCommands object.
   *
   * @param \Drupal\Core\Entity\EntityTypeManagerInterface $entityTypeManager
   *   The entity type manager.
   * @param \Drupal\Core\Extension\ModuleHandlerInterface $moduleHandler
   *   The module handler.
   * @param \Symfony\Contracts\EventDispatcher\EventDispatcherInterface $eventDispatcher
   *   The event dispatcher.
   *
   * @throws \Drupal\Component\Plugin\Exception\InvalidPluginDefinitionException
   *   Thrown if the "search_api_index" or "search_api_server" entity types'
   *   storage handlers couldn't be loaded.
   * @throws \Drupal\Component\Plugin\Exception\PluginNotFoundException
   *   Thrown if the "search_api_index" or "search_api_server" entity types are
   *   unknown.
   */
  public function __construct(EntityTypeManagerInterface $entityTypeManager, ModuleHandlerInterface $moduleHandler, EventDispatcherInterface $eventDispatcher) {
    parent::__construct();

    $this->commandHelper = new CommandHelper($entityTypeManager, $moduleHandler, $eventDispatcher, 'dt');
  }
}

```

Here is an example command (`search_api:enable`) from `Search_API`. This is an excerpt from `web/modules/contrib/search_api/src/Commands/SearchApiCommands.php`. Looking at the annotation you can see all the details of the command as well as the aliases. This annotation provides the ability to issue the command `drush search_api:enable`. The aliases allow you to use `drush sapi-en` OR `drush search-api-enable` as well. Note. You can always add aliases etc. like `drush @apc.dev sapi-en`.

```

/**
 * Enables one disabled search index.
 *
 * @param string $indexId
 *   A search index ID.
 *
 * @throws \Drupal\search_api\ConsoleException
 *   Thrown if no indexes could be loaded.
 *
 * @command search-api:enable
 *
 * @usage drush search-api:enable node_index
 *   Enable the search index with the ID node_index.
 *
 * @aliases sapi-en,search-api-enable
 */
public function enable($indexId) {
  $this->commandHelper->enableIndexCommand([$indexId]);
}

```

You also need a `composer.json` in the custom module directory. Note the `extra`, `drush`, `services` that reference the `drush.services.yml` file:

Here is the version that drush generates by default:

```
{
  "name": "org/tea_vote_cache",
  "type": "drupal-drush",
  "extra": {
    "drush": {
      "services": {
        "drush.services.yml": "^10"
      }
    }
  }
}
```

Here is the `composer.json` from a project where I used (Search API module)[https://www.drupal.org/project/search_api] module at `web/modules/contrib/search_api/composer.json`:

```
{
  "name": "drupal/search_api",
  "description": "Provides a generic framework for modules offering search capabilities.",
  "type": "drupal-module",
  "homepage": "https://www.drupal.org/project/search_api",
  "authors": [
    {
      "name": "Thomas Seidl",
      "homepage": "https://www.drupal.org/u/drunken-monkey"
    },
    {
      "name": "Nick Veenhof",
      "homepage": "https://www.drupal.org/u/nick_vh"
    },
    {
      "name": "See other contributors",
      "homepage": "https://www.drupal.org/node/790418/committers"
    }
  ],
  "support": {
    "issues": "https://www.drupal.org/project/issues/search_api",
    "irc": "irc://irc.freenode.org/drupal-search-api",
    "source": "https://git.drupalcode.org/project/search_api"
  },
  "license": "GPL-2.0-or-later",
  "require-dev": {
    "drupal/language_fallback_fix": "@dev",
    "drupal/search_api_autocomplete": "@dev"
  },
  "suggest": {
    "drupal/facets": "Adds the ability to create faceted searches.",
    "drupal/search_api_autocomplete": "Allows adding autocomplete suggestions to search fields.",
    "drupal/search_api_solr": "Adds support for using Apache Solr as a backend."
  },
  "extra": {
    "drush": {
      "services": {
        "drush.services.yml": "^9 || ^10"
      }
    }
  },
  "conflict": {
    "drupal/search_api_solr": "2.* || 3.0 || 3.1"
  }
}
```

Drush commands and parameters

You can specify the commands with a member function and all its parameters like this:

Here we have a command

```
public function commandWarmVotingCache($scope = 'current', $program_nid = 0, $options = ['option-name' => 'default']): int {
```

This expects a command like: drush cwarm current 12345

NOTE

The \$options parameter is not generally for you to use. Drush uses it for all sorts of parameters internally. It is interesting to look at the values in the debugger.

Output messages on screen

Here we use print:

```
$program_title = $program_node->getTitle();
print "Warming cache for program_nid: $program_nid Title: $program_title\n";
$progress = new Progress($program_nid);
$max_vote_number = $program_node->get('field_srp_vote_number')->value;
for ($vote_number = 0; $vote_number <= $max_vote_number; $vote_number++) {
  $status = $progress->getTeamKssCompletionStatus($vote_number);
  print "vote_number: $vote_number, Team KSS completion status: $status\n";
}
```

You can use the Drupal logger to output data to the terminal. If you specify `->info` instead of `->notice` below you will only see the output in the terminal if you add `-vvv` to the drush command i.e. `drush cwar current -vvv`

```
\Drupal::logger('tea_teks_voting')->notice("Drush cache warm requested. Scope: $scope, Program_nid = $program_nid");
```

```
if ($rc === 0) {
  $this->logger()->success(dt('Operation completed.'));
}
else {
  $this->logger()->error(dt('Failed to complete processing.'));
}
```

Drush Scripts

These are PHP Scripts that run after a full Drupal bootstrap. From https://www.drush.org/latest/commands/php_script/#options : A useful alternative to eval command when your php is lengthy or you can't be bothered to figure out bash quoting. If you plan to share a script with others, consider making a full Drush command instead, since that's more self-documenting. Drush provides command line options to the script via a variable called `$extra`.

You can execute these scripts with `drush scr` script (where script is the filename of the script to execute.). If you put the script in the docroot (or web) directory, you don't need to specify a path.

TL;DR

- 1 Put script file in docroot (No need to make it executable.)
- 2 Requires opening `<?php` tag
- 3 Execute with `drush scr` scriptname arg1 arg2
- 4 Arguments show up in the `$extra` array i.e. `$extra[0]...`
- 5 Print with `$this->output()->writeln`, `print "\n"` OR `echo "\n"`

Run your scripts

e.g. to execute `example.drush`

```
$ ddev drush php:script example.drush
```

I prefer the alias

```
$ddev drush scr example.drush
```

If you want to ssh into the DDEV container, you can use:

```
$ ddev ssh
```

```
$ drush scr example.drush
```

If you have drush installed globally, you can also run it from the host with:

```
$ drush scr example.drush
```

Arguments example

Here is a simple example `example2.drush` showing arguments:

```
<?php

use Drush\Drush;

$this->output()->writeln("Hello world!");

// $extra is an array if you issue arguments e.g.
// drush scr example2.drush abc def
$this->output()->writeln("The extra options/arguments to this command were:");
$this->output()->writeln(print_r($extra, true));
```

Here is the script being run via ddev and showing its output:

```
$ ddev drush scr example2.drush abc def
Hello world!
The extra options/arguments to this command were:
Array
(
    [0] => abc
    [1] => def
)
```

Arguments example2

Here is `opinion_renamer.php` script that expects a parameter and dies if it doesn't find it. Also if the second parameter is R, it reports that it is reverting titles.

```
<?php

print ("usage: drush scr test1 <acg term id> <R>\n");
print ("e.g. 976 is the termid for John Smith.\n");

$acg = $extra[0] ?? "";
if (empty($acg)) {
    die("no acg termid");
}
$undo = FALSE;
if (isset($extra[1])) {
    $revert = $extra[1];
    if (strtoupper($revert) == "R") {
        $undo = TRUE;
        print("reverting titles for acg $acg\n");
    }
}
```

Hello world example

Here is an example of a script from <https://raw.githubusercontent.com/drush-ops/drush/11.x/examples/helloworld.script>:

```

<?php

// This example demonstrates how to write a drush
// script. These scripts are run with the php:script command.
//
use Drush\Drush;

$this->output()->writeln("Hello world!");
$this->output()->writeln("The extra options/arguments to this command were:");
$this->output()->writeln(print_r($extra, true));

//
// We can check which site was bootstrapped via
// the '@self' alias, which is defined only if
// there is a bootstrapped site.
//
$self = Drush::aliasManager()->getSelf();
if (!$self->hasRoot()) {
    $this->output()->writeln('No bootstrapped site.');
}
else {
    $this->output()->writeln('The following site is bootstrapped:');
    $this->output()->writeln(print_r($self->legacyRecord(), true));
}

```

Executing it looks like this:

```

$ ddev drush scr example.drush
Hello world!
The extra options/arguments to this command were:
Array
(
)

The following site is bootstrapped:
Array
(
    [root] => /var/www/html/web
    [uri] => https://d9book2.ddev.site
)

```

Database query example

Here is example3.drush with a database query:

```

<?php

use Drush\Drush;

$database = \Drupal::database();
$query = $database->query("SELECT nid, vid, type FROM {node} n");
$results = $query->fetchAll();

$result_count = count($results);
$this->output()->writeln("Result count = " . $result_count);

```

entityQuery example

Here is equery1.php showing looping through results of the query

```
<?php

$query = \Drupal::entityQuery('node');
$query->condition('status', 1);
$query->condition('type', 'vote');
$query->sort('title', 'ASC');
$nids = $query->execute();

if (empty($nids)) {
  $this->output()->writeln("No votes found");
}
else {
  $this->output()->writeln("Found " . count($nids) . " articles");
}

foreach ($nids as $nid) {
  $node = \Drupal\node\Entity\Node::load($nid);
  $title = $node->getTitle();
  $this->output()->writeln("Nid: " . $nid . " title: " . $title);
}
```

You could also change values and update the nodes using code like this:

```
foreach ($nids as $nid) {
  $node = \Drupal\node\Entity\Node::load($nid);
  $title = $node->getTitle();

  $old_title = substr($title, 0, 7);
  print("$title to $old_title\n");
  $node->set('title', $old_title);
  $node->save();
}
print "finished.\n";
return;
```

Drush does that?

Who isn't impressed by the things Drush does? It really shows off the incredible talent of Moshe Weitzman and the team that keep Drush moving. Drush can do almost anything. Here are a few that I like:

List blocks

To get drush to list all the blocks on your site.

```
drush ev "print_r(array_keys(\Drupal::service('plugin.manager.block')->getDefinitions()));"
```

which outputs something like:

```
[37] => system_migrated_block
[38] => system_messages_block
[39] => system_powered_by_block
[40] => user_login_block
[41] => views_block:comments_recent-block_1
[42] => views_block:content_recent-block_1
[43] => views_block:events-block_1
[44] => views_block:related_products_and_services-block_1
[45] => views_block:who_s_new-block_1
[46] => views_block:who_s_online-who_s_online_block
[47] => views_exposed_filter_block:news_listing_for_news_landing-page_1
[48] => local_actions_block
[49] => local_tasks_block
[50] => page_title_block
[51] => broken
```

)

Generate code

Drush can write modules (maybe not quite as well as ChatGPT), but they will save you tons of time. Use drush generate module and or drush generate controller to get a nice starting point for you to write your own controllers.

For more, on generating controllers see <https://www.drush.org/latest/generators/controller/>

This is what it looks like to generate a controller:

```
$ drush generate controller

Welcome to controller generator!
-----

Module machine name [web]:
► general

Class [GeneralController]:
► ExampleController

Would you like to inject dependencies? [No]:
►

Would you like to create a route for this controller? [Yes]:
►

Route name [general.example]:
► general.book_example

Route path [/general/example]:
► /general/book_example

Route title [Example]:
► Book Example

Route permission [access content]:
►

The following directories and files have been created or updated:
-----
• /Users/selwyn/Sites/d9book2/web/modules/custom/general/general.routing.yml
• /Users/selwyn/Sites/d9book2/web/modules/custom/general/src/Controller/ExampleController.php
```

Sanitize Databases

From https://www.drush.org/latest/commands/sql_sanitize/

Sanitize the database by removing or obfuscating user data.

Commandfiles may add custom operations by implementing:

- \@hook on-event sql-sanitize-confirms. Display summary to user before confirmation.
- \@hook post-command sql-sanitize. Run queries or call APIs to perform sanitizing

Several working commandfiles may be found at <https://github.com/drush-ops/drush/tree/11.x/src/Drupal/Commands/sql>

Delete nodes

To delete all node entities in batches of 5 use:

```
drush entity:delete node --chunks=5
```

More from https://www.drush.org/latest/commands/entity_delete/

- drush entity:delete node --bundle=article. Delete all article entities.
- drush entity:delete shortcut. Delete all shortcut entities.
- drush entity:delete node 22,24. Delete nodes 22 and 24.
- drush entity:delete node --exclude=9,14,81. Delete all nodes except node 9, 14 and 81.
- drush entity:delete user. Delete all users except uid=1.
- drush entity:delete node --chunks=5. Delete all node entities in steps of 5.

Delete redirects

Delete all redirects on local site

```
drush @self entity:delete redirect
```

Watch the watchdog log

```
$ drush watchdog:tail

348 09/Jan 19:29 cron Info Cron run completed.
347 09/Jan 19:29 cron Info Execution of update_cron() took 2504.96ms.
346 09/Jan 19:29 cron Warning Attempting to re-run cron while it is already running.
345 09/Jan 19:29 cron Info Starting execution of update_cron(), execution of system_cron() took 1420.76ms.
344 09/Jan 19:29 cron Info Starting execution of system_cron(), execution of search_cron() took 123.02ms.
343 09/Jan 19:29 cron Info Starting execution of search_cron(), execution of node_cron() took 56.93ms.
342 09/Jan 19:29 cron Info Starting execution of node_cron(), execution of history_cron() took 23.6ms.
341 09/Jan 19:29 cron Info Starting execution of history_cron(), execution of file_cron() took 48.27ms.
```

Read Drupal config

To display any config value, use drush cget:

```
$ drush cget admin_toolbar.settings

menu_depth: 4
```

Or

```
$ drush cget user.settings
_core:
 default_config_hash: w314Zp7B4zzzzzzv-KwZO2E1fSSK0
langcode: en
anonymous: Anonymous
verify_mail: true
notify:
cancel_confirm: true
password_reset: true
status_activated: true
status_blocked: false
status_canceled: false
register_admin_created: true
register_no_approval_required: true
register_pending_approval: true
register: visitors_admin_approval
cancel_method: user_cancel_block
password_reset_timeout: 86400
password_strength: true
```

And yes, you can use config:set (or cset) to change configuration. For more see https://www.drush.org/latest/commands/config_get/

Check if current config matches exported config

When working on a project, to make sure you have exported any changes to config so you can check it into git, use drush cst and drush cex to export.

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

```
$ drush cex -y
```

More at https://www.drush.org/latest/commands/config_status/ and https://www.drush.org/latest/commands/config_export/

Importing the config (and overwriting what is currently in the database is accomplished with:

```
$ drush cim -y
```

User Login

```
$ drush uli 1 /admin/config (uid = 1, go directly to /admin/config)

$ drush uli --name=Sarah

$ drush uli --mail=Phase3TestPark@mightycitizen.com

$ drush @wolfman.local uli --no-browser

$ drush uli --no-browser
http://default/user/reset/1/1673815967/cLRjg5XAOIDzr5UrkirPg-kmX8KWGSwILUeyteZbsUU/login
```

Note. The URL above isn't a usable URL but you can just paste everything starting at /user onto the end of the real URL in your browser e.g.
<https://d9book2/ddev.site/> user/reset/1/1673815967/cLRjg5XAOIDzr5UrkirPg-kmX8KWGSwILUeyteZbsUU/login

```
$ drush @nicer.prod uli --no-browser --uid=11  
  
$ drush uli --no-browser --uri=team.ddev.site --mail=TestPub1@team.com  
  
$ drush uli --no-browser --uri=https://team.ddev.site --mail=arnie.williamson@team.com  
  
$ drush uli --no-browser --uri=https://d9book2.ddev.site https://d9book2.ddev.site/user/reset/1/1673816036/RMTR1AVzcR-GeBIW1ckyqJZ-1kvA3gebidmPghSjMg4/login
```

More at https://www.drush.org/latest/commands/user_login/

Drupal:directory

Return devel directory

```
$ drush dd devel  
  
/Users/selwyn/Sites/d9book2/web/modules/contrib/devel
```

Return files directory

```
$ drush dd files  
  
/Users/selwyn/Sites/d9book2/web/sites/default/files
```

Return Drupal root directory

```
$ drush dd  
  
/Users/selwyn/Sites/d9book2/web
```

Navigate to the files directory.

```
cd $(drush dd files)
```

Global Install

To install drush globally so you can just issue drush commands from anywhere on your system (mac only) use:

```
$ composer global require drush/drush:\^8.0
```

And add this to your .bashrc or .zshrc:

```
alias drush = '~.composer/vendor/drush/drush/drush'
```

Drush aliases

These allow Drush to reach out to your client websites from your local machine. The alias files should be stored in /drush/sites (that is above your docroot so they will never be publicly accessible) and can be checked into your repo. If you have a file named /drush/sites/abc.site.yml you will be able to issue drush commands like drush @abc.dev status OR drush @abc.prod sql-dump >dbprod.sql from within the directory for your abc site.

You can clear cache (cr), enable a module (en devel), dump a database to your local (sql-dump), sync files from remote to local:

```
drush @mysite cr  
drush @mysite en devel  
drush @mysite sql-dump >dbprod.sql  
drush rsync <source> <dest> (To rsync files to/from sites. More on that below)
```

These commands will only work on your computer if you have drush installed globally with all its requirements met. As usual, DDEV has a nice workaround. To use drush aliases with DDEV, you will need to setup your ssh keys to run in the containers (ddev auth ssh) and then

you can issue drush alias commands from within the containers. For example:

```
$ ddev auth ssh
$ drush @abc.test status

Drupal version : 9.4.8
Site URI      : http://abcmaterialsstg.prod.acquia-sites.com
DB driver     : mysql
DB hostname   : db-1zzzzzzzb.cdb.database.services.acquia.io
DB port       : 3306
DB username   : uMzzzzzs
DB name       : 1f3zzzzzdb
Database      : Connected
Drupal bootstrap : Successful
Default theme  : abc
Admin theme   : seven
PHP binary    : /usr/local/php8.1/bin/php
PHP config    : /usr/local/php8.1/etc/cli/php.ini
PHP OS        : Linux
PHP version   : 8.1.10
Drush script  : /var/www/html/vendor/drush/drush/drush
Drush version  : 11.3.2
Etc.
```

Example Drush alias file

Here is an example drush alias (abc.site.yml) where `abc.dev` refers to a site at the fictitious IP address of 123.45.67.123 and the `abc.devsp` refers to a site at 100.1.2.3. The first assumes there is an ssh key setup on the host whereas the second would expect a password to be entered each time a drush command is issued.

```
dev:
  host: 123.45.67.123
  user: master_gzzzzz
  root: '/home/zzzz.cloudy.com/kzzzzz/public_html/web'
  uri: 'https://doug.merrysite.com'
  ssh:
    tty: 0
  paths:
    drush-script: '/home/zzzz.cloudy.com/kzzzzz/public_html/vendor/drush/drush/drush'

devsp:
  host: 100.1.2.3
  user: spolit
  root: '/var/www/dev.paris.com/web'
  uri: 'https://dev.paris.com'
  ssh:
    tty: 0
    options: '-o PasswordAuthentication=yes'
  paths:
    drush-script: '/var/www/dev.paris.com/vendor/drush/drush/drush'
```

Example Acquia Drush alias files

Usually you will need to make your own drush alias files. Here is an example alias file from Acquia. If this file is named `/drush/sites/abc.site.yml` you will be able to issue drush commands like `drush @abc.dev status` OR `drush @abc.prod sql-dump >dbprod.sql` from within the directory for your abc site. The command `drush @abc.dev status` will return the status for the dev site. The command `drush @abc.prod sql-dump >dbprod.sql` will download the entire production from the production server database into a local file called `dbprod.sql`. For DDEV use, you should gzip the file and then you can easily import it into your local DDEV database with a command like `ddev import-db --src=dbprod.sql.gz`.

Note. The section in the file below with `ssh` and `tty` in it is to resolve a problem (from <https://github.com/drush-ops/drush/issues/4004>) with

the sql-dump drush command. Check the aforementioned Github.com issue link for side-effects.

Also, a little unusual is the prod.livedev section below. This facilitates using Acquia's live dev environment feature but is not necessary for hosting environments other than Acquia.

For more on using Drush aliases with Acquia see <https://docs.acquia.com/cloud-platform/manage/ssh/drush/aliases/>

```
# Application 'abcmaterials', environment 'dev'.
dev:
root: /var/www/html/docroot
ac-site: abcmaterials
ac-env: dev
ac-realm: prod
uri: abcmaterialsdev.prod.acquia-sites.com
host: abcmaterialsdev.ssh.prod.acquia-sites.com
user: abcmaterials.dev
paths:
drush-script: drush9
ssh:
tty: 0

# Application 'abcmaterials', environment 'test'.
test:
root: /var/www/html/docroot
ac-site: abcmaterials
ac-env: test
ac-realm: prod
uri: abcmaterialsstg.prod.acquia-sites.com
host: abcmaterialsstg.ssh.prod.acquia-sites.com
user: abcmaterials.test
paths:
drush-script: drush9
ssh:
tty: 0

# Application 'abcmaterials', environment 'prod'.
prod:
root: /var/www/html/abcmaterials.prod/docroot
ac-site: abcmaterials
ac-env: prod
ac-realm: prod
uri: abcmaterials.prod.acquia-sites.com
ssh:
tty: 0
prod.livedev:
parent: '@abcmaterials.prod'
root: /mnt/gfs/abcmaterials.prod/livedev/docroot
host: abcmaterials.ssh.prod.acquia-sites.com
user: abcmaterials.prod
paths:
drush-script: drush9
ssh:
tty: 0
```

Example Drush alias file from Drush project

You can view this on github at <https://github.com/drush-ops/drush/blob/master/examples/example.site.yml>

```
# Example of valid statements for an alias file.
```

```
# Basic Alias File Usage
#
# In its most basic form, the Drush site alias feature provides a way
# for teams to share short names that refer to the live and staging sites
# (usually remote) for a given Drupal site.
#
# 1. Make a local working clone of your Drupal site and then
#   `cd` to the project work to select it.
# 2. Add an alias file called $PROJECT/drush/sites/self.site.yml,
#   where $PROJECT is the project root (location of composer.json file).
# 3. Run remote commands against the shared live or stage sites
#
# Following these steps, a cache:rebuild on the live environment would be:
#
# $ drush @live cache:rebuild
#
# The site alias file should be named `self.site.yml` because this name is
# special, and is used to define the different environments (usually remote)
# of the current Drupal site.
#
# The contents of the alias file should look something like the example below:
#
# @code
# # File: self.site.yml
#
# live:
#   host: server.domain.com
#   user: www-admin
#   root: /other/path/to/live/drupal
#   uri: http://example.com
#
# stage:
#   host: server.domain.com
#   user: www-admin
#   root: /other/path/to/stage/drupal
#   uri: http://stage.example.com
# @endcode
#
# The top-level element names ('live' and 'stage' in the example above) are
# used to identify the different environments available for this site. These
# may be used on the command line to select a different target environment
# to operate on by prepending an '@' character, e.g. `@live` or `@stage`.
#
# All of the available aliases for a site's environments may be listed via:
#
# $ drush site:alias @self
#
# The elements of a site alias environment are:
#
# - 'host': The fully-qualified domain name of the remote system
#   hosting the Drupal instance. **Important Note: The remote-host option
#   must be omitted for local sites, as this option controls various
#   operations, such as whether or not rsync parameters are for local or
#   remote machines, and so on.
# - 'user': The username to log in as when using ssh or rsync. If each user
#   has to use own username, you can create an environment variable which holds
#   the value, and reference via ${env.PROJECT_SSH_USER} (for example). Or you may
#   omit the 'user:' item and specify a user in the `~/.ssh/config` file.
# - 'root': The Drupal root; must not be specified as a relative path.
# - 'uri': The value of --uri should always be the same as
#   when the site is being accessed from a web browser (e.g. http://example.com)
```

```

#
# Drush uses ssh to run commands on remote systems; all team members should
# install ssh keys on the target servers (e.g. via ssh-add).

# Advanced Site Alias File Usage
#
# It is also possible to create site alias files that reference other
# sites on the same local system. Site alias files for other local sites
# are usually stored in the directory `~/.drush/sites`; however, Drush does
# not search this location for alias files by default. To use this location,
# you must add the path in your Drush configuration file. For example,
# to re-add both of the default user alias path from Drush 8, put the following
# in your `~/.drush/drush.yml` configuration file:
#
# @code
# drush:
#   paths:
#     alias-path:
#       - '${env.HOME}/.drush/sites'
#       - '/etc/drush/sites'
# @endcode
#
# The command `drush core:init` will automatically configure your
# `~/.drush/drush.yml` configuration file to add `~/.drush/sites` and
# `/etc/drush/sites` as locations where alias files may be placed.
#
# A canonical alias named "example" that points to a local
# Drupal site named "http://example.com" looks like this:
#
# @code
# File: example.site.yml
# dev:
#   root: /path/to/drupal
#   uri: http://example.com
# @endcode
#
# Note that the first part of the filename (in this case "example")
# defines the name of the site alias, and the top-level key ("dev")
# defines the name of the environment.
#
# With these definitions in place, it is possible to run commands targeting
# the dev environment of the target site via:
#
# $ drush @example.dev status
#
# This command is equivalent to the longer form:
#
# $ drush --root=/path/to/drupal --uri=http://example.com status
#
# See "Additional Site Alias Options" below for more information.

# Converting Legacy Alias Files
#
# To convert legacy alias (*.aliases.drushrc.php) to yaml, run the
# site:alias-convert command.

# Altering aliases:
#
# See examples/Commands/SiteAliasAlterCommands.php for an example.

```

```
# Environment variables:
#
# Site aliases may reference environment variables, just like any Drush config
# file. For example, ${env.PROJECT_SSH_USER} will be replaced by the value
# of the PROJECT_SSH_USER environment variable.
#
# SSH site aliases may set environment variables via the 'env-vars' key.
# See below.

# Additional Site Alias Options
#
# Aliases are commonly used to define short names for
# local or remote Drupal installations; however, an alias
# is really nothing more than a collection of options.
#
# - 'docker': When specified, Drush executes via docker-compose exec rather than ssh
#   - 'service': the name of the container to run on.
#   - 'exec':
#     - 'options': Options for the exec subcommand.
# - 'os': The operating system of the remote server. Valid values
#   are 'Windows' and 'Linux'. Be sure to set this value for all remote
#   aliases because the default value is PHP_OS if 'remote-host'
#   is not set, and 'Linux' (or $options['remote-os']) if it is. Therefore,
#   if you set a 'remote-host' value, and your remote OS is Windows, if you
#   do not set the 'OS' value, it will default to 'Linux' and could cause
#   unintended consequences, particularly when running 'drush sql-sync'.
# - 'ssh': Contains settings used to control how ssh commands are generated
#   when running remote commands.
#   - 'options': Contains additional commandline options for the ssh command
#     itself, e.g. "-p 100"
#   - 'tty': Usually, Drush will decide whether or not to create a tty (via
#     the ssh '-t' option) based on whether the local Drush command is running
#     interactively or not. To force Drush to always or never create a tty,
#     set the 'ssh.tty' option to 'true' or 'false', respectively.
# - 'paths': An array of aliases for common rsync targets.
# Relative aliases are always taken from the Drupal root.
# - 'files': Path to 'files' directory. This will be looked up if not
#   specified.
# - 'drush-script': Path to the remote Drush command.
# - 'command': These options will only be set if the alias
#   is used with the specified command. In the example below, the option
#   '--no-dump' will be selected whenever the @stage alias
#   is used in any of the following ways:
#   - `drush @stage sql-sync @self @live`
#   - `drush sql-sync @stage @live`
#   - `drush sql-sync @live @stage`
# NOTE: Setting boolean options broke with Symfony 3. This will be fixed
#   in a future release. See: https://github.com/drush-ops/drush/issues/2956
# - 'env-vars': An array of key / value pairs that will be set as environment
#   variables.
#
# Complex example:
#
# @code
# # File: remote.site.yml
#
# live:
#   host: server.domain.com
#   user: www-admin
#   root: /other/path/to/drupal
```

```
# uri: http://example.com
# ssh:
#   options: '-p 100'
# paths:
#   drush-script: '/path/to/drush'
# env-vars:
#   PATH: /bin:/usr/bin:/home/www-admin/.composer/vendor/bin
#   DRUPAL_ENV: live
# command:
#   site:
#     install:
#       options:
#         admin-password: 'secret-secret'
# @endcode

# Site Alias Files for Service Providers
#
# There are a number of service providers that manage Drupal sites as a
# service. Drush allows service providers to create collections of site alias
# files to reference all of the sites available to a single user. In order
# to do this, a new location must be defined in your Drush configuration
# file:
#
# @code
# drush:
#   paths:
#     alias-path:
#       - '${env.HOME}/.drush/sites/provider-name'
# @endcode
#
# Site aliases stored in this directory may then be referenced by its
# full alias name, including its location, e.g.:
#
# $ drush @provider-name.example.dev
#
# Such alias files may still be referenced by their shorter name, e.g.
# `@example.dev`. Note that it is necessary to individually list every
# location where site alias files may be stored; Drush never does recursive
# (deep) directory searches for alias files.
#
# The `site:alias` command may also be used to list all of the sites and
# environments in a given location, e.g.:
#
# $ drush site:alias @provider-name
#
# Add the option `--format=list` to show only the names of each site and
# environment without also showing the values in each alias record.

# Wildcard Aliases for Service Providers
#
# Some service providers that manage Drupal sites allow customers to create
# multiple "environments" for a site. It is common for these providers to
# also have a feature to automatically create Drush aliases for all of a
# user's sites. Rather than write one record for every environment in that
# site, it is also possible to write a single "wildcard" alias that represents
# all possible environments. This is possible if the contents of each
# environment alias are identical save for the name of the environment in
# one or more values. The variable `${env-name}` will be substituted with the
# environment name wherever it appears.
```

```

#
# Example wildcard record:
#
# @code
## File: remote-example.site.yml
# *:
# host: ${env-name}.server.domain.com
# user: www-admin
# root: /path/to/${env-name}
# uri: http://${env-name}.remote-example.com
# @endcode
#
# With a wildcard record, any environment name may be used, and will always
# match. This is not desirable in instances where the specified environment
# does not exist (e.g. if the user made a typo). An alias alter hook in a
# policy file may be used to catch these mistakes and report an error.
# @see SiteAliasAlterCommands for an example on how to do this.

# Developer Information
#
# See https://github.com/consolidation/site-alias for more developer
# information about Site Aliases.
#
# An example appears below. Edit to suit and remove the @code / @endcode and
# leading hashes to enable.
#
# @code
## File: mysite.site.yml
# local:
# This environment is an example of the DockerCompose transport.
# docker:
#   service: drupal
#   exec:
#     options: --user USER
# stage:
#   uri: http://stage.example.com
#   root: /path/to/remote/drupal/root
#   host: mystagingserver.myisp.com
#   user: publisher
#   os: Linux
#   paths:
#     - files: sites/mydrupalsite.com/files
#     - custom: /my/custom/path
#   command:
#     sql:
#       sync:
#         options:
#           no-dump: true
# dev:
#   root: /path/to/docroot
#   uri: https://dev.example.com
# @endcode

# Example of rsync with exclude-paths
#
# Note that most options typically passed to rsync via `drush rsync` are
# "passthrough options", which is to say they appear after the `--` separator
# on the command line. Passthrough options are actually arguments, and
# it is not possible to set default arguments in an alias record. The
# `drush rsync` command does support two options, `--mode` and `--exclude-paths`.

```

```
# which are interpreted directly by Drush. Default values for these options
# may be specified in an alias record, as shown below.

#
# @code

# dev:
#   root: /path/to/docroot
#   uri: https://dev.example.com
#   command:
#     core:
#       rsync:
#         options:
#           mode: rlpz
#           exclude-paths: 'css:imagecache:ctools:js:tmp:php:styles'
#   @endcode
```

Drush rsync

You can use aliases to rsync files using drush rsync. The syntax is: drush rsync <source> <dest> or more specifically: drush rsync @sitename.prod:/path/to/files /path/to/local/destination

You can use @self to represent the local site e.g. drush rsync @labs.prod:%files @self:%files

To copy everything in ./keys folder to the keys folder on the host use:

```
$ drush rsync ./keys/ @sf:/home/master/applications/dir/public_html/keys/
```

Copy 1 file to the dir test server:

```
drush rsync dbprod.sql.gz @dir.testsp:/var/www/dir.tater.com/web/
```

You can also use :%files as a source or dest. Examples below:

```
drush rsync @fightclub.prod:/var/www/html/fightclub.prod/docroot/sites/default/files/ ~/Sites/fightclub/docroot/sites/default/files/
```

```
drush rsync @dak.prodsp:%files /Users/selwyn/Sites/dakbackup/web/sites/default/files/
```

```
drush rsync @dak.prodsp:/var/www/dak.tater.com/web/ /Users/selwyn/Sites/dakbackup/web/
```

Copy 1 file from local to @dak.dev: drush rsync dbprod-good.sql.gz
@dak.dev:/home/1234567.cloudwaysapps.com/hzzzz/public_html/drush-backups/

Drush SQL-sync

To sync databases use: drush sql-sync <source> <destination>

e.g.

```
$ drush sql-sync @flexy.stage @flex.dev
```

SQL Queries

Examples of SQL queries. Note. you can update, delete or select.

```
$ drush sqlq "select nid, vid, type from node where type = 'vote'"
35 40 vote
36 41 vote
```

```
$ drush sqlq "select count(*) as redirects"
1
```

Drush launcher

install drush launcher from <https://github.com/drush-ops/drush-launcher>

By installing the drush launcher globally on your local machine, you can simply type drush on the command line, and the launcher will find and execute the project specific version of drush located in your project's vendor directory.

from <https://github.com/drush-ops/drush-launcher>: In order to avoid dependency issues, it is best to require Drush on a per-project basis via Composer (`$ composer require drush/drush`). This makes Drush available to your project by placing it at `vendor/bin/drush`.

However, it is inconvenient to type `vendor/bin/drush` to execute Drush commands. By installing the drush launcher globally on your local machine, you can simply type drush on the command line, and the launcher will find and execute the project specific version of drush located in your project's vendor directory.

Drupal 7 Drush scripts

Here are some old Drupal 7 Drush scripts in case they are of any interest

```
//content created my admin - d7
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
FROM {node} n WHERE n.uid = :uid', array(':uid' => $uid));
// Result is returned as a iterable object that returns a stdClass object on each iteration
foreach ($result as $record) {
  // Perform operations on $record->title, etc. here.
  // in this example the available data would be mapped to object properties:
  // $record->nid, $record->title, $record->created
  print_r($record);
}
```

```
// Login as admin
$admin_uid = 1;
$form_state = array('uid' => $admin_uid);
user_login_submit(array(), $form_state);
// Now the logged in user global $user object become available.
global $user;
print_r($user);
```

```

// Args
drush_print("Hello world!");
drush_print();
drush_print("The arguments to this command were:");
//
// If called with --everything, use drush_get_arguments
// to print the commandline arguments. Note that this
// call will include 'php-script' (the drush command)
// and the path to this script.
//
if (drush_get_option('everything')) {
  drush_print(" " . implode("\n ", drush_get_arguments()));
}
//
// If --everything is not included, then use
// drush_shift to pull off the arguments one at
// a time. drush_shift only returns the user
// commandline arguments, and does not include
// the drush command or the path to this script.
//
else {
  while ($arg = drush_shift()) {
    drush_print(' ' . $arg);
  }
}

drush_print();

```

It is optional to have <?php opening tag and a #! lines:

This script takes a filename as a parameter and checks if the file exists.

```

<?php
#!/Users/selwyn/.composer/vendor/bin/drush

drush_print("Hello world!");
drush_print();
$arg = drush_shift();
if ($arg) {
  if (file_exists($arg)) {
    drush_print("yep - it's here");
  }
}

```

Resources

- Drush docs site <https://www.drush.org/latest/>
- Drush repository <https://github.com/drush-ops/drush>
- Drush code generators <https://www.drush.org/latest/generators/all/>
- Stack Exchange Drush issues <https://drupal.stackexchange.com/questions/tagged/drush>
- Drush issue queue on github <https://github.com/drush-ops/drush/issues>
- Creating Drush 9 commands and porting legacy commands October 2017 by Paweł Ginalski <https://gbyte.dev/blog/creating-drush-9-commands-and-porting-legacy-commands>
- Porting commands to Drush 9 from Moshe Weitzman September 2017 <https://weitzman.github.io/blog/port-to-drush9>

- Drush alias usage on Acquia <https://docs.acquia.com/cloud-platform/manage/ssh/drush/aliases/>
 - Controlling multiple sites with Drush 9 from [Joachim Noreiko](http://www.noreiko.com/blog/controlling-multiple-sites-drush-9) Mar 2019 <http://www.noreiko.com/blog/controlling-multiple-sites-drush-9>
-
-

[Back to top](#)

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

Page last modified: Apr 24 2023.

[Edit this page on GitHub](#)

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

Email

TABLE OF CONTENTS

- [Send email](#)
- [Using tokens in hook_mail](#)
- [Reference](#)

views 170

Send email

Function to send an email in your module with a hook_mail() to set the parameters.

```
use Drupal\Core\Mail\MailManagerInterface;
use Drupal\Core\Language\LanguageManagerInterface;

/**
 * Function to send an email in your module.
 *
 * @param string $to
 *   The recipient of the email.
 * @param array $params
 *   An array of parameters to use in the email.
 * @param string $subject
 *   The subject of the email.
 * @param string $body
 *   The body of the email.
 *
 * @return bool
 *   TRUE if the email was sent successfully, FALSE otherwise.
 */
function send_my_email($to, $params, $subject, $body) {
  // Get the language manager service.
  $language_manager = \Drupal::service('language_manager');

  // Get the default language.
  $default_language = $language_manager->getDefaultLanguage()->getId();

  // Get the mail manager service.
  $mail_manager = \Drupal::service('plugin.manager.mail');

  // Set up the email parameters.
  $module = 'my_module';
  $key = 'my_email_key';

  // Get the email headers.
  $headers = [
    'Content-Type' => 'text/html; charset=UTF-8; format=flowed; delsp=yes',
    'From' => 'sender@example.com',
    'Reply-To' => 'reply-to@example.com',
  ];

  // Set up the email body.
  $body_params = [
```

```

'message' => $params['message'],
'name' => $params['name'],
};

// Send the email.

$result = $mail_manager->mail($module, $key, $to, $default_language, $body_params, NULL, TRUE, $headers);

// Return the result.

return $result['result'];
}

/**
 * Implements hook_mail().
 */
function my_module_mail($key, &$message, $params) {
switch ($key) {
case 'my_email_key':
$message['subject'] = t('My Email Subject');
$message['body'][] = $params['message'];
$message['body'][] = t('From: @name', ['@name' => $params['name']]);
break;
}
}

```

Using tokens in hook_mail

Here is an example in a hook_mail call where tokens are used:

```

/**
 * Implements hook_mail().
 */
function hello_world_mail($key, &$message, $params) {
switch ($key) {
case 'hello_world_log':
$message['from'] = \Drupal::config('system.site')->get('mail');
$message['subject'] = t('There is an error on your website');
$message['body'][] = $params['message'];
if (isset($params['user'])) {
$user_message = 'The user that was logged in: [current-user:name];
$message['body'][] = \Drupal::token()->replace($user_message, ['current-user' => $params['user']]);
}
break;
}
}

```

Reference

- [Sending html mails in Drupal 8/9 programmatically An example Drupal module including Twig template by Joris Snoek - August 2020](#)
 - [Sending Emails Using OOP and Dependency Injection in Drupal 8, 9 By Alex Novak - November 2020.](#)
 - [How email works in Drupal - updated July 2021](#)
 - [Sendgrid Integration Drupal module](#)
-
-

[Back to top](#)

Page last modified: Jun 10 2023.

[Edit this page on GitHub](#)

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

Entities

TABLE OF CONTENTS

- [Overview](#)
- [Config entity types](#)
- [Content entity types](#)
- [Query an entity by title and type](#)
- [Create an entity](#)
- [Save an entity](#)
- [Create article node entity with attached image](#)
- [Update a node entity and add some terms](#)
- [Get the entity type and content type \(or bundle type\)](#)
- [Identify entities](#)
- [Create a file entity](#)
- [Entity Validation](#)
- [Resources](#)

views 176

Overview

The Entity API is used for manipulating entities (CRUD: create, read, update, delete). Entity validation has its own API (which could validate an Entity saved via REST, rather than a form, for example).

Entities come in two flavors: Content and Config(uration). The data storage mechanism moved from being field-centric in Drupal 7 to entity-centric in Drupal 8, 9 and 10. This implies that all fields attached to an entity share the same storage backend, making querying a lot easier. Entity types are registered with Drupal as plugins.

While Drupal supports custom entities, I haven't found a need for them in my projects. In most of my experience, just using real nodes (sometimes unpublished) has sufficed admirably.

Config entity types

They store configuration information e.g: views, imagestyles, roles, NodeType (which define node bundles).

- They are not revisionable
- They don't have fields/are not fieldable.
- They don't support entity translation interface(TranslatableInterface), but can still be translated using config's translation API.

Content entity types

e.g. comment, user, taxonomy term, node and now media (bundles for media are file, image, audio, video, remote video etc.)

more at <https://www.drupal.org/docs/drupal-apis/entity-api/introduction-to-entity-api-in-drupal-8>

Query an entity by title and type

This example counts the number of entities of type `article` with the name `$name`. Note that access checking must be [explicitly specified on content entity queries](#).

```
$query = $this->nodeStorage->getQuery()
->accessCheck(FALSE)
->condition('type', 'article')
->condition('title', $name)
->count();

$count_nodes = $query->execute();

if ($count_nodes == 0) {
```

Create an entity

To create a new entity object, use the `entity_create`. **NOTE** that this only creates an entity object and does not persist it.

```
$node = entity_create('node', array(
  'title' => 'New Article',
  'body' => 'Article body',
  'type' => 'article',
));
```

If you know what the entity class name (bundle type) is, you can use it directly.

```
$node = Node::create(array(
  'title' => 'New Article',
  'body' => 'Article body',
  'type' => 'article',
));
```

Save an entity

Entity save is done by calling the instance's `save` method.

```
$node->save();
```

Create article node entity with attached image

```
use Drupal\node\Entity\Node;

$data = file_get_contents('https://www.drupal.org/files/druplicon-small.png');
$file = file_save_data($data, 'public://druplicon.png', FILE_EXISTS_RENAME);

$node = Node::create([
  'type'      => 'article',
  'title'     => 'A new article',
  'field_image' => [
    'target_id' => $file->id(),
    'alt' => 'Drupal',
    'title' => 'Drupal logo'
  ],
]);
assert($node->isNew(), TRUE);
$node->save();
assert($node->isNew(), FALSE);
```

Update a node entity and add some terms

```

use Drupal\node\Entity\Node;
use Drupal\taxonomy\Entity\Term;

$node = Node::load(4);
$term1 = Term::load(1);
$term2 = Term::load(2);
$node->field_tags->setValue([$term1, $term2]);
$node->save();

```

Get the entity type and content type (or bundle type)

```

use Drupal\Core\Entity\EntityInterface;

function hook_entity_presave(EntityInterface $entity) {
  // getEntityType() returns 'node'.
  // $entity->bundle() returns 'article'.
  if ($entity->getEntityType() == 'node' && $entity->bundle() == 'article') {
    // Do your stuff here
  }
}

```

Identify entities

from <https://www.drupal.org/docs/drupal-apis/entity-api/working-with-the-entity-api>

```

// Make sure that an object is an entity.
if ($entity instanceof \Drupal\Core\Entity\EntityInterface) {
}

// Make sure it's a content entity.
if ($entity instanceof \Drupal\Core\Entity\ContentEntityInterface) {
}
// or:
if ($entity->getEntityType()->getGroup() == 'content') {
}

// Get the entity type or the entity type ID.
$entity->getEntityType();
$entity->getEntityTypeId();

// Make sure it's a node.
if ($entity instanceof \Drupal\node\NodeInterface) {
}

// Using entityType() works better when the needed entity type is dynamic.
$needed_type = 'node';
if ($entity->getEntityType() == $needed_type) {
}

```

Create a file entity

Note. Drupal is ok with files existing (for example in sites/default/files) without file entities but it probably makes more sense to have file entities connected with those files. When you create file entities, Drupal tracks the files in the tables: file_managed and file_usage. You can also create media entities where bundle='file'.

Once the file already exists, I write the file entity. I do need a mime type

```

switch (strtoupper($type)){
  case 'PDF':
    $mimetype = 'application/pdf';
    break;
  case 'JPG':
    $mimetype = 'image/jpeg';
    break;
  case 'POW':
    $mimetype = 'application/vnd.ms-powerpoint';
    break;
}

// Create file entity.
$file = File::create();
$file->setFileUri($destination);
$file->setOwnerId(1);
$file->setMimeType($mimetype);
$file->setFileName($this->fileSystem->basename($final_destination));
$file->setPermanent();
$file->save();

$fid = $file->id();
return $fid;

```

Entity Validation

This looks quite interesting although I haven't had an opportunity to try it yet.

From Entity Validation API overview at <https://www.drupal.org/node/2015613>

The [entity validation API](#) should be used to validate values on a node, like any other content entity. While that was already possible to define validation constraints for base fields and for a field type, there was no API to add constraints to a specific field. This has been addressed.

```

/**
 * Implements hook_entity_bundle_field_info_alter().
 */
function mymodule_entity_bundle_field_info_alter(&$fields, \Drupal\Core\Entity\EntityTypeInterface $entity_type, $bundle) {
  if ($entity_type->id() == 'node' && !empty($fields['myfield'])) {
    $fields['myfield']->setPropertyConstraints('value', [
      'Range' => [
        'min' => 0,
        'max' => 32,
      ],
    ]);
  }
}

```

Custom validation constraints can also be defined. These links may be useful: [ForumLeafConstraint](#) and [ForumLeafConstraintValidator](#)

Also from <https://drupalize.me/tutorial/entity-validation-api?p=2792> (you need a paid membership to read the whole tutorial): Drupal includes the [Symfony Validator component](#), and provides an Entity Validation API to assist in validating the values of fields in an entity. By using the Entity Validation API you can ensure that your validation logic is applied to Entity CRUD operations regardless of how they are triggered. Whether editing an Entity via a Form API form, or creating a new Entity via the REST API, the same validation code will be used.

Resources

- [Entity API on Drupal.org updated Jan 2021](#)
- [Introduction to Drupal Entity API from Drupal.org updated Sep 2022](#)
- [Drupal entity API cheat sheet Updated July 2018](#)

- [Drupal 8 Entity query API cheat sheet by Keith Dechant , Software Architect - updated February 2021](#)
 - [Entity Validation API tutorial from https://drupalize.me](#) (requires paid membership to read the whole tutorial) updated February 2022 at
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) 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.

Forms, Form API and AJAX

TABLE OF CONTENTS

- [Overview](#)
- [Find a form id in the page source](#)
- [Add buttons to your custom forms](#)
- [Modify a button on a form with hook_form_alter](#)
- [Hide a field with hook_form_alter](#)
- [Hide revision info and moderation state](#)
- [Multiple fields on the same controller/page](#)
- [Conditional fields and field states API \(#states\)](#)
 - [Conditional fields in a form](#)
 - [Conditional fields in node add or edit form](#)
- [Get the key and value from a select drop-down](#)
- [Autocomplete](#)
 - [Add an autocomplete taxonomy field](#)
 - [Add a views-driven entity autocomplete field](#)
 - [Disable autocomplete for user login and password fields](#)
- [Validating input](#)
 - [Validate string length](#)
 - [Validate an email](#)
 - [Validate date](#)
 - [Validate a node add or edit](#)
- [Displaying Forms](#)
 - [Embedding a form:](#)
 - [Show a form in a block](#)
 - [Provide a block template for a form in a block](#)
- [Redirecting](#)
 - [Form submission with redirect](#)
 - [Ajax redirect](#)
 - [AJAX redirect from a select element \(dropdown\)](#)
- [Add Javascript to a form](#)
- [AJAX Forms](#)
 - [Popup an AJAX modal dialog](#)
 - [AJAX modal dialog with redirect example](#)
 - [Ajax submit](#)
 - [Ajax redirect](#)
 - [AJAX redirect from a select element \(dropdown\)](#)
 - [Update a value in another field\(I am I want\) using AJAX](#)
 - [I Am I Want revisited](#)
 - [Custom responses](#)
 - [How do you find all the possible AJAX commands to use with addCommand\(\)?](#)
 - [Another AJAX Submit example](#)
- [Config Forms](#)
 - [Generate a config form with drush](#)
 - [Config forms overview](#)
- [The basics of implementing forms](#)

- [Location](#)
- [Base Classes for forms](#)
- [Create your form class by extending Formbase](#)
- [The main methods](#)
 - [getFormId\(\)](#)
 - [buildForm\(\)](#)
 - [submitForm\(\)](#)
- [Form validation example #1](#)
- [Form Validation example #2](#)
- [Field attributes](#)
- [Form Elements](#)
- [Retrieving field values](#)
- [Resources](#)

views 227

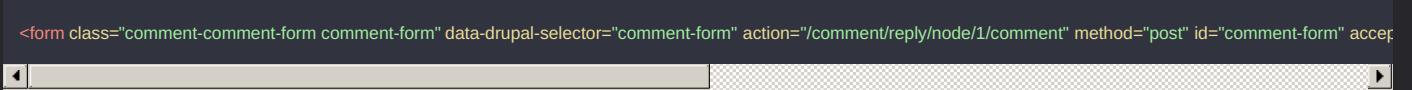
Overview

Forms can act just like controllers. You can set up a route and call the form in a very similar fashion. See the example route below:

```
batch_examples.batch:
path: '/batch-examples/batchform'
defaults:
  _title: 'Batch'
  _form: 'Drupal\batch_examples\Form\BatchForm'
requirements:
  _permission: 'access content'
```

Find a form id in the page source

When you need to make changes to a form, it can take a little time to find the form. You often need to find the form id as the first step. To find the form_id for a node comment form, start by editing an article node with the comment form displaying. Inspect code in chrome and look for something like this:



```
<form class="comment-comment-form comment-form" data-drupal-selector="comment-form" action="/comment/reply/node/1/comment" method="post" id="comment-form" accept-
```

The formid is comment_comment_form. Note *dashes will need to become underscores in your code*.

Alternatively, you can add a `hook_form_alter` and `print_r` or `dsm` the `$form_id`. If you prefer, you could also log it to the watchdog log:

```
function nisto_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id){
  \Drupal::messenger()->addMessage('Form ID: ' . $form_id);
  // Or log it to watchdog log.
  \Drupal::logger('nisto_form_alter')->notice('Form ID: ' . $form_id);
```

Notice that a node add form looks like `node_catastrophe_notice_form` while a node edit form looks more like this: `node_catastrophe_notice_edit_form`

Add buttons to your custom forms

Many forms need extra buttons. Fortunately Drupal allows this to be done easily. In the code below there is a submit button which will execute the `submitForm` function. There are two additional buttons: `update_nodes` which (on click) calls the `updateNodes` function and `cache_warmer` which (one click) will execute the `warmCaches` function.

```

$form['sanity_fieldset']['actions'] = [
  '#type' => 'actions',
];
$form['sanity_fieldset']['actions']['submit'] = [
  '#type' => 'submit',
  '#value' => $this->t('Submit'),
];
}

$form['sanity_fieldset']['actions']['update_nodes'] = [
  '#type' => 'submit',
  '#value' => $this->t('Update'),
  '#submit' => ['::updateNodes'],
];
}

$form['sanity_fieldset']['actions']['cache_warmer'] = [
  '#type' => 'submit',
  '#value' => $this->t('Warm Caches'),
  '#submit' => ['::warmCaches'],
];

```

and here is the beginning of the `submit` function that goes along with the `update_nodes` submit button:

```

public function updateNodesFromCache(array &$form, FormStateInterface $form_state) {
  $program_nid = $form_state->get('program_nid');
  $vote_number = $form_state->get('vote_number');
  $program_node = $form_state->get('program');
  $program_title = $program_node->getTitle();
  $team_nid = $program_node->get('field_srp_team_ref')[\$vote_number]->target_id;

  $chunk_size = 20;
  $operations = [];

  $cache_id = "expectations.program.$program_nid.vote.$vote_number.team.$team_nid";
  $cache_data = \Drupal::cache()->get($cache_id);
  if ($cache_data) {
    $expectations = $cache_data->data;
    $chunks = array_chunk($expectations, $chunk_size);
    for ($i = 0; $i < count($chunks); $i++) {
      $operations[] = [
        '\Drupal\tea_teks_voting\Batch\VotingCacheBatch::updateExpectations',
        [$chunks[$i]],
      ];
    }
  }

  $batch = [
    'title' => $this->t("Updating nodes from voting caches for $program_title"),
    'progress_message' => $this->t('Completed @current out of @total chunks.'),
    'finished' => '\Drupal\tea_teks_voting\Batch\VotingCacheBatch::batchFinished',
    'operations' => $operations,
  ];
  batch_set($batch);
}

```

Note. The example above shows a batch function. You can read more in the [Batch and Queue chapter](#)

Modify a button on a form with hook_form_alter

in a .module file like web/modules/custom/mymod/mymod.module we can change the caption on the "save" button to "Comment"

```

function mymod_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id) {
  if ($form_id == 'comment_comment_form') {
    $form['actions']['submit']['#value'] = t('Comment');
  }
}

```

Here is a dropdown form element that I added an item to:

```
$form['topic']['#options'][992] = "selwyn";
```

And in another .module file I nuke the contents of the dropdown topic and reload it with different content. These are children of term with tid 2806 sorted by the term name:

```

function nisto_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id){
  // xdebug_break();
  if ($form_id == "views_exposed_form") {
    // Clear the current options.
    $form["topic"]["#options"] = [];

    $tid = 2806;
    $terms = \Drupal::entityTypeManager()->getStorage('taxonomy_term')->loadChildren($tid);
    foreach($terms as $term) {
      $childTerm = $term->get('tid')->value;
      $termName = ucfirst(strtolower($term->get("name")->value));
      $tid = $term->get("tid")->value;
      $form["topic"]["#options"][$tid] = $termName;
    }
    natcasesort($form["topic"]["#options"]);
  }
}

```

Hide a field with hook_form_alter

In a .module file, I used this code to remove access to these fields:

```

$form['field_highlight_section']['#access'] = 0;
$form['field_accordion_section']['#access'] = 0;

```

This grays out a field:

```
$form['field_text2']['#disabled'] = true;
```

Here is the whole function where I also check what these is currently in use:

```

function dan_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id) {
  $activeThemeName = \Drupal::service('theme.manager')->getActiveTheme();
  if ($activeThemeName->getName() == 'seven') {
    $form['#attributes']['novalidate'] = 'novalidate';
  }
  if ($form_id == 'views-exposed-form-site-search-page-search') {

    $form['#theme'] = ['header_search_form'];

  }
  // User is editing or adding content of type overview.
  if ($form_id == 'node_overview_edit_form' || $form_id == 'node_overview_form') {
    \Drupal::messenger()->addMessage("blah");
    $form['field_text2']['#disabled'] = true;
    $form['field_highlight_section']['#access'] = 0;
    $form['field_accordion_section']['#access'] = 0;
  }
}

```

Hide revision info and moderation state

In a .module file you can turn off (or hide) revision information and moderation state like this.

```

$form['revision_information']['#access'] = FALSE;
$form['moderation_state']['#access'] = FALSE;

```

Multiple fields on the same controller/page

If you need to have the same form appear multiple times on a page, you need to add a little special logic. In one example, I had several items displayed on a page, and each one needed an option to add feedback by the user. This required the use of a static class variable to uniquely identify each instance of the form on the page.

```

/**
 * Class SrpAddFeedbackForm.
 */
class SrpAddFeedbackForm extends FormBase {

  /**
   * @var int $instanceId
   * Used to make sure the getFormId is always unique.
   */
  private static int $instanceId;
  ...
}

```

In the `getFormId()` method where you would normally just use `return 'tea_teks_srp_feedback_add';`, you can use the following code instead:

```

/**
 * {@inheritDoc}
 */
public function getFormId() {
  if (empty(self::$instanceId)) {
    self::$instanceId = 1;
  }
  else {
    self::$instanceId++;
  }
  return 'tea_teks_srp_feedback_add' . self::$instanceId;
}

```

Conditional fields and field states API (#states)

There are some nice articles at <https://mushtaq.ch/blog/11/drupal-8-conditionally-hide-a-form-field> and

<https://www.lullabot.com/articles/form-api-states> that go into more detail.

Note. There is a very workable conditional fields module https://www.drupal.org/project/conditional_fields which lets you do the same sort of thing without any code.

The magic sauce is to use the jQuery selector to identify the field that will control the states. You can see the left side of the => has the jQuery code to select a checkbox or radio button.

```
:input[name="copies_yes_no"] => [value] => '0'
```

Conditional fields in a form

When creating a form, follow these steps:

- 1 Create the field that will control the other field (radio buttons in this case)
- 2 Create the field that will be hidden (or manipulated in various ways)
- 3 Create the field with a ['#states'] index

Create the copies_yes_no field, then the how_many_oversized field and then finally the #states attribute which will make the how_many_oversized field appear if you click Yes to the copies_yes_no field.

Here is an Example form which will only show the how_many_oversized field when the copies_yes_no radio is set to yes:

```
$form['nonstandard']['copies_yes_no'] = [
  '#type' => 'radios',
  '#title' => $this->t('Did the requestor ask for copies of nonstandard documents (e.g., oversized paper, DVD, or VHS tape)?'),
  '#default_value' => 0,
  '#options' => [
    0 => $this->t('Yes'),
    1 => $this->t('No'),
  ],
];

$form['nonstandard']['how_many_oversized'] = [
  '#type' => 'number',
  '#title' => $this->t('Oversized paper copies (e.g., 11 inches by 17 inches, greenbar, bluebar), # of pages'),
  '#min' => 0,
  '#max' => 65535,
  '#size' => 6,
];

$form['nonstandard']['how_many_oversized']['#states'] = [
  // Only show when copies_yes_no is yes.
  'visible' => [
    ':input[name="copies_yes_no"]' => [value] => '0',
  ],
];
```

Conditional fields in node add or edit form

To customize how fields are handled in an existing form, use hook_form_alter and check to see if you are a node add or edit form for our type of node. Set the state to visible for field_cn_original_notice (so the user can see it) if the field_cn_extension has a value of checked=TRUE. This has the effect of dynamically displaying the field once the checkbox for field_cn_extension is clicked. The rest of the function has various other field tweaks

This is from a .module file

```

function org_mods_form_alter(array &$form, FormStateInterface $form_state, $form_id) {
  $accountProxy = \Drupal::currentUser();
  $account = $accountProxy->getAccount();
  if (($form_id == 'node_catastrophe_notice_form') || ($form_id == 'node_catastrophe_notice_edit_form')) {

    //Disable field.
    $form['field_cn_end_date']['widget'][0]['value']['#attributes']['disabled'] = 'disabled';
    //Or the simpler.
    $form['field_cn_end_date']['#disabled'] = TRUE;

    if (!$account->hasPermission('administer catastrophe notice')) {
      $form['field_cn_notes']['#access'] = FALSE;
      $form['field_cn_initial_start_date']['#access'] = FALSE;
      $form['field_cn_initial_end_date']['#access'] = FALSE;
      $form['revision_information']['#access'] = FALSE;
      $form['moderation_state']['#access'] = FALSE;
    }

    // Hides Title since it is automatically populated on presave.
    $form['title']['#access'] = FALSE;

    // When the value of field_cn_extension is checked, show this field.
    $form['field_cn_original_notice']['#states'] = [
      'visible' => [
        ':input[name="field_cn_extension[value]"]' => ['checked' => TRUE],
      ],
    ];
  }
}

```

Here is a variant on this theme for making a field visible or required. In this example, it isn't using the `[value]` as part of the name. When I tried that above, it didn't seem to work. Perhaps I didn't have the correct jquery selector.

```

$form['field_bla_bla']['#states']= [
  // Only show when scoring_unavailable is not checked.
  'visible' => [
    ':input[name="scoring_unavailable"]' => ['unchecked' => TRUE],
  ],
  'required' => [
    ':input[name="scoring_unavailable"]' => ['unchecked' => TRUE],
  ],
];

```

Get the key and value from a select drop-down

To get the key and the value that the user sees in the dropdown use the following. When the dropdown was created, we gave it array of strings so the key is a zero-based number and the value (`$educationLevel`) is the string. This shows how to get both.

```

$key = $form_state->getValue('education_level');
$educationLevel = $form['education_level']['#options'][$key];

```

Autocomplete

Add an autocomplete taxonomy field

This makes a field on your form that automatically starts populating with terms when you start typing. Here the `$vid` is a vocabulary machine name like `media_tags`. Not sure what `#tags` does – It doesn't seem to be required. Notice vocab id (`vid`) is the taxonomy machine name not a number.

```

$vid1 = 'media_tags';
$form['tags'] = [
  '#type' => 'entity_autocomplete',
  '#title' => $this->t('Tags'),
  '#target_type' => 'taxonomy_term',
  '#selection_settings' => [
    'target_bundles' => [$vid1], //could be [$vid1, $vid2..],
  ],
  // '#tags' => TRUE,
];

```

Add a views-driven entity autocomplete field

This allows you to create a field on your form with a user field. This will be an autocomplete field which uses the view: `users_view` and the display `users`. It allows you to start typing a username in the field and all matching users will be displayed in the dropdown below the field:

```

$form['user'] = [
  '#type' => 'entity_autocomplete',
  '#target_type' => 'user',
  '#selection_handler' => 'views',
  '#selection_settings' => [
    'view' => [
      'view_name' => 'users_view',
      'display_name' => 'users',
      'arguments' => []
    ],
    'match_operator' => 'CONTAINS'
  ],
];

```

from <https://drupal.stackexchange.com/questions/308870/entity-autocomplete-form-api-field-with-viewsselection-handler>

Disable autocomplete for user login and password fields

In a .module file use the following code.

```

/**
 * Implements hook_form_FORM_ID_alter().
 *
 * Turn off autocomplete on login.
 */
function drir_form_user_login_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id) {
  $form['pass'][#attributes]['autocomplete'] = 'off';
  $form['name'][#attributes]['autocomplete'] = 'off';
}

function drir_form_user_pass_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id) {
  $form['name'][#attributes]['autocomplete'] = 'off';
}

```

Validating input

Validate string length

Check a string length for the `company_name` field.

```

public function validateForm(array &$form, FormStateInterface $formState) {
  if (!$formState->isEmpty('company_name')) {
    if (strlen($formState->getValue('company_name')) <= 5) {
      //Set validation error.
      $formState->setErrorByName('company_name', t('Company name is less than 5 characters'));
    }
  }
}

```

Validate an email

From web/modules/custom/rsvp/src/Form/RSVPForm.php we call Drupal's `email.validator` service and if it fails, `setErrorByName()`

```

public function validateForm(array &$form, FormStateInterface $form_state) {
  $value = $form_state->getValue('email');
  if (!\Drupal::service('email.validator')->isValid($value)) {
    $form_state->setErrorByName('email', t('The email %mail is not valid.', ['%mail' => $value]));
  }

  parent::validateForm($form, $form_state);
}

```

Validate date

You can also add a custom validation in a .module file. Here we use `setTime()`to remove the time part of a datetime so we can make comparisons of just the date.

From a .module file.

```

/**
 * Custom form validation for catastrophe notices.
 *
 * Checks if a user enters a date that is more than
 * 2 days earlier than the current date, but only if they didn't
 * check the extension checkbox.
 *
 * @param $form
 * @param \Drupal\Core\Form\FormStateInterface $form_state
 */
function cn_form_validate($form, FormStateInterface $form_state) {
  $extension = $form_state->getValue('field_cn_extension');
  if (is_array($extension)) {
    $extension = $extension['value'];
  }
  $start_date = $form_state->getValue('field_cn_start_date');
  if ($start_date) {
    $start_date = $start_date[0]['value'];
    $start_date->setTime(0, 0, 0);
    $now = new Drupal\Core\Datetime\DrupalDateTime();
    $now->modify("-2 days");
    $now->setTime(0, 0, 0);

    if ($start_date < $now && !$extension) {
      $form_state->setErrorByName('field_cn_start_date', t('The starting date is more than 2 days in the past. Please select a more recent date.'));
    }
  }
}

```

Validate a node add or edit

In org_mods.module we implement a `hook_form_alter`, and add a validate callback function for anonymous users only.

```
/**  
 * Implements hook_form_alter().  
 */  
  
function org_mods_form_alter(array &$form, FormStateInterface $form_state, $form_id) {  
  $accountProxy = \Drupal::currentUser();  
  $account = $accountProxy->getAccount();  
  
  // Add special validation for anonymous users (node add) only.  
  if (($accountProxy->isAnonymous() && ($form_id == 'node_catastrophe_notice_form'))) {  
    $form['#validate'][] = 'cn_form_validate';  
  }  
}
```

And here is the custom validate function which does some fun date arithmetic.

```
/**  
 * Custom form validation for catastrophe notices.  
 *  
 * Checks if a user enters a date that is more than  
 * 2 days earlier than the current date, but only if they didn't  
 * check the extension checkbox.  
 *  
 * @param $form  
 * @param \Drupal\Core\Form\FormStateInterface $form_state  
 */  
  
function cn_form_validate($form, FormStateInterface $form_state) {  
  $extension = $form_state->getValue('field_cn_extension');  
  if (is_array($extension)) {  
    $extension = $extension['value'];  
  }  
  $start_date = $form_state->getValue('field_cn_start_date');  
  if ($start_date) {  
    $start_date = $start_date[0]['value'];  
    $start_date->setTime(0, 0, 0);  
    $now = new Drupal\Core\Datetime\DrupalDateTime();  
    $now->modify("-2 days");  
    $now->setTime(0, 0, 0);  
  
    if ($start_date < $now && !$extension) {  
      $form_state->setErrorByName('field_cn_start_date', t('The starting date is more than 2 days in the past. Please select a later date'));  
    }  
  }  
}
```

Displaying Forms

Embedding a form:

In this example, there is a form called `ExampleForm` at `web/modules/custom/test/src/Form/ExampleForm.php`.

To render a form programmatically, either inside a Controller or a block, use the `FormBuilder` service. The form builder can be injected using the `form_builder` service key or used statically to then build the form (which returns a render array)

```
$form = \Drupal::formBuilder()->getForm('Drupal\test\Form\ExampleForm');  
$build['egform'] = $form;  
return $build;
```

Show a form in a block

In dev1/web/modules/custom/rsvp/src/Plugin/Block/RSVPBlock.php you can see in the build() method, we invoke a form like this:

```
class RSVPBlock extends BlockBase {

    /**
     * @inheritDoc
     */
    public function build() {
        return \Drupal::formBuilder()->getForm('Drupal\rsvp\Form\RSVPForm');
    }
}
```

In docroot/modules/custom/quick_pivot/src/Plugin/Block/QuickPivotSubscribeBlock.php we use dependency injection to pass in the FormBuilderInterface and then get the form in a very similar way.

The constructor grabbed the formbuilder like this:

```
public function __construct(array $configuration, $plugin_id, $plugin_definition, ConfigFactoryInterface $config_factory, FormBuilderInterface $form_builder) {
    parent::__construct($configuration, $plugin_id, $plugin_definition);

    $this->configFactory = $config_factory;
    $this->formBuilder = $form_builder;
}

public function build() {
    return $this->formBuilder->getForm('Drupal\quick_pivot\Form\QuickPivotSubscribeForm');
}
```

Provide a block template for a form in a block

In /modules/custom/dan_pagination/src/Form/VideoPaginationForm.php I have a form which is displayed in a block. The usual block template file provided by the theme is block.html.twig and looks like this:

(Here is an image of this source code. Strangely, Jekyll/Github requires me to jump through some hoops for TWIG source. I'm still experimenting.)

```
<div{{ attributes }}>
  {{ title_prefix }}
  {% if label %}
    <h2{{ title_attributes }}>{{ label }}</h2>
  {% endif %}
  {{ title_suffix }}
  {% block content %}
    {{ content }}
  {% endblock %}
</div>
```

Here is the source:

```

<div
{{ attributes }}>
{{ title_prefix }}
{% if label %}
<h2{{ title_attributes }}>{{ label }}</h2>
{% endif %}
{{ title_suffix }}
{{ block content %}}
{{ content }}
{{ endblock %}}
</div>

```

The template outputs the guts of the block as

```
 {{ block content }}
```

For my custom theme called dprime, I added a new template file at `themes/custom/dprime/templates/block/block--videopaginationblock.html.twig` and added lots of fun stuff to output the form in bits and pieces.

e.g. like here, to display the previous_clip item from the form's render array which looks like this:

```

$form['previous_clip'] = [
  '#type' => 'markup',
  '#markup' => "(Clip $previous_clip_num/$clip_count)",
];

```

And in the template, you can see `content.previous_clip` referencing this content.

```

<div class="cell medium-6 medium-order-2 text-center">

  {{ content.select }}

</div>
<div class="cell small-6 medium-order-1 medium-3">
  <a href="" class="linkPager linkPager--prev">
    <span class="linkPager-icon">
      <span class="icon--arrowLeft" data-grunticon-embed=""></span>
    </span>
    <span class="linkPager-text">
      <span class="linkPager-title">Prev</span>
      {{ content.previous_clip }}
    </span>
  </a>
</div>

```

Redirecting

Form submission with redirect

Here is a `submitForm()` function from `docroot/modules/custom/websphere_commerce/modules/checkout/src/Form/ReviewForm.php`

The call to `$form_state->getValues()` retrieves all the values in the form. The rest of the logic checks a value and redirects the user to a specific page.

```

public function submitForm(array &$form, FormStateInterface $form_state) {
  $values = $form_state->getValues();

  if ($values['op'] == 'Goback') {
    redirectUser('/checkout' . $values['cart_id'] . '/billing');
  }
}

```

Here is the source for the redirectUser() call from above.

```

function redirectUser($path, $route = FALSE) {
  if (!$route) {
    $redirectUrl = Url::fromUserInput($path)->toString();
    $response = new RedirectResponse($redirectUrl);
    $response->send();
  }
  return;
}

```

Ajax redirect

If you want to redirect to the /cart url, you must add an AJAX command. See the [RedirectCommand in the API Reference](#)

```

$cartUrl = Url::fromUri('internal:/cart');

$ajax_response->addCommand(
  new RedirectCommand($cartUrl->toString())//Note this is a string!!
);
return $ajax_response;

```

In a non-ajax form, to redirect to the cart url, we would just use something like this:

```
$form_state->setRedirectUrl($cartUrl);
```

AJAX redirect from a select element (dropdown)

Here I set up a dropdown with the url's and when the user makes a change in the dropdown, the browser goes to that url. The url's are /node/1 /node/2 etc. For the correct url to be built, we have to prefix "internal:" to them and that happens in the callback function mySelectChange().

```

/**
 * {@inheritDoc}
 */
public function buildForm(array $form, FormStateInterface $form_state, $nojs = NULL) {

  // Get the form values and raw input (unvalidated values).
  $values = $form_state->getValues();

  // Define a wrapper id to populate new content into.
  $ajax_wrapper = 'my-ajax-wrapper';

  // Select element.
  $form['my_select'] = [
    '#type' => 'select',
    '#empty_value' => '',
    '#empty_option' => '- Select a value -',
    '#default_value' => (isset($values['my_select']) ? $values['my_select'] : ''),
    '#options' => [
      '/node/1' => 'One',
      '/node/2' => 'Two',
      '/node/3' => 'Three'
    ]
}

```

```

    ],
    '#ajax' => [
      'callback' => [$this, 'mySelectChange'],
      'event' => 'change',
      'wrapper' => $ajax_wrapper,
    ],
  ];
}

// Build a wrapper for the ajax response.
$form['my_ajax_container'] = [
  '#type' => 'container',
  '#attributes' => [
    'id' => $ajax_wrapper,
  ]
];

return $form;
}

/**
 * The callback function for when the `my_select` element is changed.
 */
public function mySelectChange(array $form, FormStateInterface $form_state) {

  $values = $form_state->getValues();

  $response = new AjaxResponse();
// $url = Url::fromUri('internal:/node/2');
$url = Url::fromUri('internal:' . $values['my_select']);

$command = new RedirectCommand($url->toString());
$response->addCommand($command);
return $response;
}

//Don't forget an empty submitForm().
public function submitForm(array &$form, FormStateInterface $form_state) {
  // This function left blank intentionally.
}

```

Invoke the form from the controller with:

```

return
return \Drupal::formBuilder()->getForm('Drupal\org_opinions\Form\IndividualOpinionForm');
```

The form is at `docroot/modules/custom/org_opinions/src/Form/IndividualOpinionForm.php`.

And don't forget these:

```

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Link;
use Drupal\Core\Url;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\RedirectCommand;
```

Let's say you have several select elements on the form like `my_select` and `my_select2` like this (Sorry - not too creative, I know.):

```

$ajax_wrapper = 'my-ajax-wrapper';
// Select.
$form['my_select'] = [
  '#type' => 'select',
  '#empty_value' => '',
  '#empty_option' => '- Select a value -',
  '#default_value' => ($isset($values['my_select'])) ? $values['my_select'] : ''),
  '#options' => [
    '/node/1' => 'One',
    '/node/2' => 'Two',
    '/node/3' => 'Three'
  ],
  '#ajax' => [
    'callback' => [$this, 'mySelectChange'],
    'event' => 'change',
    'wrapper' => $ajax_wrapper,
  ],
];
$form['my_select2'] = [
  '#type' => 'select',
  '#empty_value' => '',
  '#empty_option' => '- Select a value -',
  '#default_value' => ($isset($values['my_select'])) ? $values['my_select'] : ''),
  '#options' => [
    '/node/4' => 'Four',
    '/node/5' => 'Five',
    '/node/6' => 'Six'
  ],
  '#ajax' => [
    'callback' => [$this, 'mySelectChange'],
    'event' => 'change',
    'wrapper' => $ajax_wrapper,
  ],
];

```

Both use the same callback: mySelectChange. We can make the callback a little smarter by figuring out internally which element called it.

```

/**
 * Callback function for changes to the `my_select` any select element.
 */
public function mySelectChange(array $form, FormStateInterface $form_state) {
  $values = $form_state->getValues();

  // $elem stores the element info
  $elem = $form_state->getTriggeringElement();
  // $value[$elem["#name"]]] stores the path like /node/2

  $response = new AjaxResponse();
  // Internal URLs must look like this: 'internal:/node/2'.
  // $url = Url::fromUri('internal:' . $values['my_select']);
  $url = Url::fromUri('internal:' . $values[$elem["#name"]]);

  $command = new RedirectCommand($url->toString());
  $response->addCommand($command);

  return $response;
}

```

Add Javascript to a form

This code is in the [examples](#) module in the `DependentDropdown` example where one field depends on the value from another

From `web/modules/contrib/examples/ajax_example/src/Form/DependentDropdown.php` the form has some Javascript included via a library:

```
public function buildForm(array $form, FormStateInterface $form_state, $nojs = NULL) {
  // Add our CSS and tiny JS to hide things when they should be hidden.
  $form['#attached']['library'][] = 'ajax_example/ajax_example.library';
```

The `ajax_example.libraries.yml` looks like this:

```
ajax_example.library:
  version: 1.x
  css:
    base:
      css/ajax-example-base.css: {}
  js:
    js/ajax-example.js: {}
```

Notice in the `buildForm()` function that the code references the machine name for the library (not the library's filename which is `ajax_example.libraries.yml`.)

Here is the js for completeness:

```
(function ($) {

  // Re-enable form elements that are disabled for non-ajax situations.
  Drupal.behaviors.enableFormItemsForAjaxForms = {
    attach: function () {
      // If ajax is enabled, we want to hide items that are marked as hidden in
      // our example.
      if (Drupal.ajax) {
        $('.ajax-example-hide').hide();
      }
    }
  };

})(jQuery);
```

AJAX Forms

When adding ajax to a form, you will need the code:

```
$form['#attached']['library'][] = 'core/drupal.dialog.ajax';
```

This line attaches the `core/drupal.dialog.ajax` library to the form and is necessary to render the modal dialogs. Alternatively, you can include this as a dependency in your module's `*.info.yml` file.

There are some really sweet [writeups about AJAX forms](#) and [AJAX Dialog boxes](#)

Popup an AJAX modal dialog

If you want to have a form pop up a modal dialog or do something via ajax you have to do some slightly special stuff.

First define what will appear on the dialog

```

$checkoutLink = '/checkout' . $get_order_item_id . '/shipping';

$success_modal_popup = [
  '#theme' => 'add_cart_success_modal_popup',
  '#data' => [
    'product_title' => $productDetails->title->value . $popup_title,
    'checkout_link' => $checkoutLink,
    'cart_link' => '/cart',
    'continue_shopping_link' => '<span class="continue-shopping>"Continue Shopping</p>',
    'product_id'=>$product_id,
    'product_price'=>number_format($productDetails_p->price[1]->value, 2),
    'product_category'=>'Singer',
    'product_quantity'=>$product_qty,
  ],
];
$content['#markup'] = render($success_modal_popup);
$content['#attached']['library'][] = 'core/drupal.dialog.ajax';
$content['#attached']['library'][] = 'websphere_commerce_cart/minicart';

```

Then add an ajax command to open the dialog:

```

$ajax_response->addCommand(
  new OpenModalDialogCommand(t($popup_title), $content, ['width' => '60%', 'dialogClass' => 'product-cart-popup'])
);
return $ajax_response;

```

Don't forget to **return** the `$ajax_response`;

Here is a slightly example displaying a modal dialog:

```

$addTocartFailed = $websphere_config->get('cart.add_to_cart_failed');

$success_modal_popup = [
  '#theme' => 'add_cart_success_modal_popup',
  '#data' => [
    'addtocart_failed' => $addTocartFailed,
    'product_id'=>$product_id,
    'product_price'=>number_format($productDetails_p->price[1]->value, 2),
    'product_category'=>'Singer',
    'product_quantity'=>$product_qty,
  ],
];
$content['#markup'] = render($success_modal_popup);
$content['#attached']['library'][] = 'core/drupal.dialog.ajax';
$ajax_response->addCommand(
  new OpenModalDialogCommand(t($popup_title), $content, ['width' => '60%', 'dialogClass' => 'product-cart-popup'])
);
return $ajax_response;

```

AJAX modal dialog with redirect example

From docroot/modules/custom/websphere_commerce/modules/product/src/Form/AddToCartForm.php.

```

use Drupal\Core\Ajax\RedirectCommand;

class AddToCartForm extends FormBase {
  // Create constructor and create functions for dependency injection

  public function __construct(PrivateTempStoreFactory $temp_store_factory, SessionManagerInterface $session_manager, AccountInterface $current_user) {
    $this->tempStoreFactory = $temp_store_factory;
    $this->sessionManager = $session_manager;
    $this->currentUser = $current_user;
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('user.private_tempstore'), $container->get('session_manager'), $container->get('current_user')
    );
  }

  // Use buildForm() function to create the elements on the form:
  public function buildForm(array $form, FormStateInterface $form_state, $nid = NULL, $productId = NULL) {

```

AJAX SUBMIT

Submit element is a little special. See all that '#ajax' stuff?

```

$form['submit'] = [
  '#type' => 'submit',
  '#attributes' => ['class' => ['mobile-hide']],
  '#id' => 'add_to_cart',
  '#value' => $this->t('Add to cart'),
  '#button_type' => 'primary',
  '#ajax' => [
    'callback' => '::add_to_cart_submit',
    'event' => 'click',
    'progress' => [
      'type' => 'throbber',
      'wrapper' => 'editor-settings-wrapper',
    ],
  ],
];
return $form;

```

In this implementation, there is an empty `submitForm()` function. For the ajax submit callback we use `add_to_cart_submit()`. Note how a new `AjaxResponse` is created.

```

public function add_to_cart_submit(array &$form, FormStateInterface $form_state) {
  global $user_status;
  $inputs = $form_state->getUserInput();
  $ajax_response = new AjaxResponse();
  $product_qty = $inputs['product_qty'];

```

Note. This should probably be a static function to avoid this symfony error:

```
TypeError: Argument 1 passed to Drupal\Core\Routing\RequestContext::fromRequest() must be an instance of Symfony\Component\HttpFoundation\Request, null given
```

AJAX REDIRECT

If you want to redirect to the `/cart` url, you must add an AJAX command. See the [RedirectCommand in the API Reference](#)

```

$cartUrl = Url::fromUri('internal:/cart');

$ajax_response->addCommand(
  new RedirectCommand($cartUrl->toString()) //Note this is a string!!
);

return $ajax_response;

```

In a non-ajax form, to redirect to the cart url, we would just use something like this:

```
$form_state->setRedirectUrl($cartUrl);
```

AJAX redirect from a select element (dropdown)

Here I set up a dropdown with the url's and when the user makes a change in the dropdown, the browser goes to that url. The url's are /node/1 /node/2 etc. For the correct url to be built, we have to prefix "internal:" to them and that happens in the callback function mySelectChange().

```

/**
 * {@inheritDoc}
 */

public function buildForm(array $form, FormStateInterface $form_state, $nojs = NULL) {

  // Get the form values and raw input (unvalidated values).
  $values = $form_state->getValues();

  // Define a wrapper id to populate new content into.
  $ajax_wrapper = 'my-ajax-wrapper';

  // Select element.
  $form['my_select'] = [
    '#type' => 'select',
    '#empty_value' => '',
    '#empty_option' => '- Select a value -',
    '#default_value' => (isset($values['my_select']) ? $values['my_select'] : ''),
    '#options' => [
      '/node/1' => 'One',
      '/node/2' => 'Two',
      '/node/3' => 'Three'
    ],
    '#ajax' => [
      'callback' => [$this, 'mySelectChange'],
      'event' => 'change',
      'wrapper' => $ajax_wrapper,
    ],
  ];

  // Build a wrapper for the ajax response.
  $form['my_ajax_container'] = [
    '#type' => 'container',
    '#attributes' => [
      'id' => $ajax_wrapper,
    ]
  ];

  return $form;
}

/**
 * The callback function for when the `my_select` element is changed.
 */

```

```

public function mySelectChange(array $form, FormStateInterface $form_state) {

  $values = $form_state->getValues();

  $response = new AjaxResponse();
  // $url = Url::fromUri('internal:/node/2');
  $url = Url::fromUri('internal:' . $values['my_select']);

  $command = new RedirectCommand($url->toString());
  $response->addCommand($command);
  return $response;
}

//Don't forget an empty submitForm().
public function submitForm(array &$form, FormStateInterface $form_state) {
  // This function left blank intentionally.
}

```

Invoke the form from the controller with:

```
return
```

```
return \Drupal::formBuilder()->getForm('Drupal\org_opinions\Form\IndividualOpinionForm');
```

The form is at docroot/modules/custom/org_opinions/src/Form/IndividualOpinionForm.php.

And don't forget these:

```

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Link;
use Drupal\Core\Url;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\RedirectCommand;

```

Let's say you have several select elements on the form like `my_select` and `my_select2` like this (Sorry - not too creative, I know.):

```

$ajax_wrapper = 'my-ajax-wrapper';
// Select.
$form['my_select'] = [
  '#type' => 'select',
  '#empty_value' => '',
  '#empty_option' => '- Select a value -',
  '#default_value' => ($isset($values['my_select'])) ? $values['my_select'] : ''),
  '#options' => [
    '/node/1' => 'One',
    '/node/2' => 'Two',
    '/node/3' => 'Three'
  ],
  '#ajax' => [
    'callback' => [$this, 'mySelectChange'],
    'event' => 'change',
    'wrapper' => $ajax_wrapper,
  ],
];
$form['my_select2'] = [
  '#type' => 'select',
  '#empty_value' => '',
  '#empty_option' => '- Select a value -',
  '#default_value' => ($isset($values['my_select'])) ? $values['my_select'] : ''),
  '#options' => [
    '/node/4' => 'Four',
    '/node/5' => 'Five',
    '/node/6' => 'Six'
  ],
  '#ajax' => [
    'callback' => [$this, 'mySelectChange'],
    'event' => 'change',
    'wrapper' => $ajax_wrapper,
  ],
];

```

Both use the same callback: mySelectChange. We can make the callback a little smarter by figuring out internally which element called it.

```

/**
 * Callback function for changes to the `my_select` any select element.
 */
public function mySelectChange(array $form, FormStateInterface $form_state) {
  $values = $form_state->getValues();

  // $elem stores the element info
  $elem = $form_state->getTriggeringElement();
  // $value[$elem["#name"]]] stores the path like /node/2

  $response = new AjaxResponse();
  // Internal URLs must look like this: 'internal:/node/2'.
  // $url = Url::fromUri('internal' . $values['my_select']);
  $url = Url::fromUri('internal' . $values[$elem["#name"]]);

  $command = new RedirectCommand($url->toString());
  $response->addCommand($command);
  return $response;
}

```

Update a value in another field(I am I want) using AJAX

This was used for a web page that had 2 dropdown fields. It showed I am a ____ and I want ____.

The code makes the assumption that there is only 1 matching I want for each I am which is way too limiting but it does illuminate the techniques somewhat. Therefor, if you select an I am, you only get 1 choice for an I want.

There is nothing too interesting in the `lamiwantBlock.php` but in the `lamiwantForm.php` it gets a little more juicy. The form presents two dropdown fields, I am and I want and a go button. The value in the I am dropdown controls the value in the I want.

From: txg/web/modules/custom/lamiwant/src/Form/lamiwantForm.php.

The code loads \$nids from the database, loops thru them, putting them into an array \$iam indexed by node id. Then it defines the `$form['iam']` element with all the good #ajax stuff including:

- 1 The '#submit' callback: `::submitSelectIam`
- 2 The `#executes_submit_callback => TRUE`
- 3 The 'callback' => '`::ajaxReplaceIwantForm`' specifies what function to call when this field is "submitted"
- 4 The 'wrapper' => 'iwant-container' is the container that will get replaced
- 5 The 'method' => 'replace' which I guess means replace everything in said wrapper.

Then it defines the `$form['want_container']` and the `$form['want_container']['iwant']` element in readiness for the AJAX magic. Lastly it defines the submit go button.

```

public function buildForm(array $form, FormStateInterface $form_state) {
  $nodeStorage = $this->entityTypeManager->getStorage('node');
  // Get i_am_i_want node ids.
  $nids = $nodeStorage->getQuery()
    ->accessCheck(TRUE)
    ->condition('field_iam', NULL, 'IS NOT NULL')
    ->condition('field_iwant', NULL, 'IS NOT NULL')
    ->condition('field_link', NULL, 'IS NOT NULL')
    ->condition('status', NodeInterface::PUBLISHED)
    ->execute();
  $nodes = $nodeStorage->loadMultiple($nids);
  $iam = [];
  foreach ($nodes as $node) {
    /* @var \Drupal\node\NodeInterface $node */
    $iam[$node->id()] = $node->get('field_iam')->getString();
  }
  $form['iam'] = [
    '#type' => 'select',
    '#title' => $this->t('I am'),
    '#options' => $iam,
    '#required' => TRUE,
    '#submit' => ['::submitSelectIam'],
    '#executes_submit_callback' => TRUE,
    '#ajax' => [
      'callback' => '::ajaxReplaceIwantForm',
      'wrapper' => 'iwant-container',
      'method' => 'replace',
    ],
  ];
  $iwant = [];
  // Updates iwant values according to iam.
  if ($node = $form_state->get('node')) {
    $iwant[] = $node->get('field_iwant')->getString();
  }
  $form['iwant_container'] = [
    '#type' => 'container',
    '#prefix' => '<div id="iwant-container">',
    '#suffix' => '</div>',
  ];
  $form['iwant_container']['iwant'] = [
    '#type' => 'select',
    '#title' => $this->t('and I want to'),
    '#options' => $iwant,
  ];
  $form['actions'] = [
    '#type' => 'actions',
  ];
  $form['actions']['submit'] = [
    '#type' => 'submit',
    '#value' => $this->t('Go'),
  ];
  return $form;
}

```

Below is the callback `submitSelectIam` (referenced above) that gets called when the `iam` field value changes. The code grabs the `iam` value from the form, loads up that node and stores it in the `$form_state` with `$form_state->set('node', $node)`; for use later to decide where to jump to.

In `submitForm()`, it grabs the node, pulls out the URI and does a `setRedirectUrl()` which causes a jump to that URL.

In submitSelectIam(), it loads whichever value the user has put in the iam select. Then he stores that node in the \$form_state, then calls \$form_state->setRebuild(). This is the code that runs each time iam is chosen.

```
/**  
 * Handles submit call when iam field is selected.  
 */  
  
public function submitSelectIam(array $form, FormStateInterface $form_state) {  
  $iam = $form_state->getValue('iam');  
  $node = $this->entityTypeManager->getStorage('node')->load($iam);  
  $form_state->set('node', $node);  
  $form_state->setRebuild();  
}
```

Then in the callback for the ajax iam element. It returns whatever needs to be replaced in the wrapper container (from above). It simply returns the new value for that container (wrapper).

```
'#ajax' => [  
  'callback' => '::ajaxReplaceWantForm',  
  'wrapper' => 'iwant-container',  
  'method' => 'replace',
```

Then, add the ajax callback.

```
/**  
 * Handles switching the iam selector.  
 */  
  
public function ajaxReplaceWantForm($form, FormStateInterface $form_state) {  
  return $form['iwant_container'];  
}
```

And finally, this code does the actual redirecting when the user clicks go:

```
public function submitForm(array &$form, FormStateInterface $form_state) {  
  /* @var \Drupal\node\NodeInterface $node */  
  $node = $form_state->get('node');  
  $uri = $node->get('field_link')->getString();  
  $url = Url::fromUri($uri);  
  $form_state->setRedirectUrl($url);  
}
```

This version handles external URL's

```
public function submitForm(array &$form, FormStateInterface $form_state) {  
  /* @var \Drupal\node\NodeInterface $node */  
  $node = $form_state->get('node');  
  // Get redirect uri from node.  
  $fieldLink = $node->get('field_link')->getString();  
  $url = Url::fromUri($fieldLink);  
  if (UrlHelper::isExternal($fieldLink)) {  
    $response = new TrustedRedirectResponse($url->toString());  
    $form_state->setResponse($response);  
  }  
  else {  
    $form_state->setRedirectUrl($url);  
  }  
}
```

When this runs, it has the I am drop down showing ‘select’ and the I want dropdown is empty. The I want dropdown is unresponsive. Hmm.

When I debug, as soon as I change the I am value, buildForm() runs, then submitSelectIam(), then buildForm() again and then ajaxReplaceIwantForm(). On subsequent runs, it skips the first buildForm() and runs through the sequence of

- 1 submitSelectIam()
- 2 buildForm()
- 3 ajaxReplaceIwantForm()

In submitSelectIam() this code \$form_state->set('node', \$node); stores the node into the form state for later use. You can ->set() anything into a form for use later. This means that the buildForm can pull it back out to figure out the value for the I want field, like this (in buildForm()). Note. You can do this in a submit callback but not a callback defined as '#ajax'. For some reason, they are ignored if you do that type.

```
// Updates iwant values according to iam.  
if ($node = $form_state->get('node')) {  
  $iwant[] = $node->get('field_iwant')->getString();  
}
```

This is the magic bullet as it limits the \$iwant[] output to only 1 value and also provides the value needed by the submit (go) button so it knows where to send the output. Unfortunately this just sets the \$iwant[] array to the only one possible iwant value. That is not as useful as one would want.

I Am I Want revisited

Revisiting this code to make it limit the iwant field to only valid values:

From ddev81/web/modules/custom/iamiwant/src/Form/IamiwantForm.php here is buildForm() .

```
public function buildForm(array $form, FormStateInterface $form_state) {  
  $nodeStorage = $this->entityTypeManager->getStorage('node');  
  // Get i_am_i_want node ids.  
  $nids = $nodeStorage->getQuery()  
    ->accessCheck(TRUE)  
    ->condition('field_iam', NULL, 'IS NOT NULL')  
    ->condition('field_iwant', NULL, 'IS NOT NULL')  
    ->condition('field_link', NULL, 'IS NOT NULL')  
    ->condition('status', NodeInterface::PUBLISHED)  
    ->execute();  
  
  // Display the message if nodes don't exist.  
  if (empty($nids)) {  
    $form['error'] = [  
      '#type' => 'item',  
      '#markup' => $this->t('No "I am I want" nodes found.'),  
    ];  
  }  
  
  $nodes = $nodeStorage->loadMultiple($nids);  
  $iam = [];  
  // Build I am select values.  
  foreach ($nodes as $node) {  
    $iam[$node->id()] = $node->get('field_iam')->getString();  
  }  
  $form['iam'] = [  
    '#type' => 'select',  
    '#title' => $this->t('I am'),  
    '#options' => array_unique($iam),  
    '#required' => TRUE,  
    '#submit' => ['::submitSelectIam'],  
    '#executes_submit_callback' => TRUE,  
    '#ajax' => [  
      'callback' => '::ajaxReplaceIwantForm',  
      'wrapper' => 'iwant-container',  
    ]  
  ];
```

```

'method' => 'replace',
],
];
$iwant = [];

// Updates iwant values according to iam.
$node = $form_state->get('node');
if ($node) {
  $iwant[] = $node->get('field_iwant')->getString();
}

$iwantarray = $form_state->get('iwantarray');
if ($iwantarray) {
  $iwant = [];
  foreach ($iwantarray as $key => $value) {
    $iwant[$key] = $value;
  }
}

$form['iwant_container'] = [
  '#type' => 'container',
  '#prefix' => '<div id="iwant-container">',
  '#suffix' => '</div>',
];
$form['iwant_container']['iwant'] = [
  '#type' => 'select',
  '#title' => $this->t('and I want to'),
  '#options' => $iwant,
  '#empty_value' => '',
  '#empty_option' => '- Select -',
  '#prefix' => '<div class="cell imw-want">',
  '#suffix' => '</div>',
  '#submit' => ['::submitSelectIwant'],
  '#executes_submit_callback' => TRUE,
  '#ajax' => [
    'callback' => '::ajaxUpdateActionsForm',
    'wrapper' => 'actions-container',
    'method' => 'replace',
    'progress' => [
      'type' => 'none',
    ],
  ],
];
;

// Go button.
$form['actions'] = [
  '#type' => 'actions',
  '#prefix' => '<div class="cell imw-submit" id="actions-container">',
  '#suffix' => '</div>',
];
$form['actions']['submit'] = [
  '#type' => 'submit',
  '#value' => $this->t('Go'),
];
;

if (!$form_state->get('node')) {
  $form['actions']['submit']['#attributes']['disabled'] = 'disabled';
}
;

return $form;
}

```

Notice that we are checking the `$iwantarray` variable which is set in the `submitSelectIam()` as shown below. I pull the value from the iam select

dropdown, look up all the possible values for iwant and populate them in an indexed array. It is indexed by nid which is important later.

```
/**  
 * Handles submit call when iam field is selected.  
 */  
  
public function submitSelectIam(array $form, FormStateInterface $form_state) {  
  $nodeStorage = $this->entityTypeManager->getStorage('node');  
  $iam = $form_state->getValue('iam');  
  $node = $nodeStorage->load($iam);  
  $iam_text = $node->get('field_iam')->value;  
  // Rebuild the iwant values  
  // Get i_am_i_want node ids.  
  $nids = $nodeStorage->getQuery()  
    ->accessCheck(TRUE)  
    ->condition('type', 'i_am_i_want')  
    ->condition('field_iam', $iam_text)  
    ->condition('field_iwant', NULL, 'IS NOT NULL')  
    ->condition('field_link', NULL, 'IS NOT NULL')  
    ->condition('status', NodeInterface::PUBLISHED)  
    ->execute();  
  $nodes = $nodeStorage->loadMultiple($nids);  
  foreach ($nodes as $node) {  
    $iwant[$node->id()] = $node->get('field_iwant')->getString();  
  }  
  
  $form_state->set('iwantrarray', $iwant);  
  $form_state->setRebuild();  
}
```

Here is the callback for when the iwant is selected. Most importantly, the line which sets the \$node into the form_state (\$formstate->set('node', \$node)) stores the node, so the submit function can figure out where the go button should take us.

```
/**  
 * Handles submit call when iwant field is selected.  
 */  
  
public function submitSelectIwant(array $form, FormStateInterface $form_state) {  
  $nodeStorage = $this->entityTypeManager->getStorage('node');  
  $iwant = $form_state->getValue('iwant');  
  $node = $nodeStorage->load($iwant);  
  $form_state->set('node', $node);  
  $form_state->setRebuild();  
}
```

These two little chaps handle updating the iwant container and the go button (enabling it when there is a valid node set.). These get fired after the buildForm() is run so they can extract meaningful info from the nice render array that is built and update just the container they are pointed at. How do I know which container (or wrapper) that is? See below..

```

/**
 * Fired from the iam selection to update the iwant select field.
 */
public function ajaxReplaceIwantForm($form, FormStateInterface $form_state) {
  return $form['iwant_container'];
}

/**
 * Fired from the iwant selection to update the go button.
 */
public function ajaxUpdateActionsForm($form, FormStateInterface $form_state) {
  return $form['actions'];
}

```

In the buildForm, we specified '#ajax' and identified a callback function with a wrapper – this says “replace” the ‘iwant-container’ by calling the ajaxReplaceIwantForm() function.

```

$form['iam'] = [
  '#type' => 'select',
  '#title' => $this->t('I am'),
  '#options' => array_unique($iam),
  '#required' => TRUE,
  '#submit' => '::submitSelectIam',
  '#executes_submit_callback' => TRUE,
  '#ajax' => [
    'callback' => '::ajaxReplaceIwantForm',
    'wrapper' => 'iwant-container',
    'method' => 'replace',
  ],
];

```

And if you look at the definition of the form in buildForm, you'll see that the \$form['iwant_container'] is clearly defined like this:

```

$form['iwant_container'] = [
  '#type' => 'container',
  '#prefix' => '<div id="iwant-container">',
  '#suffix' => '</div>',
];

```

With a form element in it

```

$form['iwant_container']['iwant'] = [
  '#type' => 'select',
  '#title' => $this->t('and I want to'),
  '#options' => $iwant,
  '#empty_value' => '',
  '#empty_option' => '- Select -',
  '#prefix' => '<div class="cell imw-want">',
  '#suffix' => '</div>',
  '#submit' => ['::submitSelectWant'],
  '#executes_submit_callback' => TRUE,
  '#ajax' => [
    'callback' => '::ajaxUpdateActionsForm',
    'wrapper' => 'actions-container',
    'method' => 'replace',
    'progress' => [
      'type' => 'none',
    ],
    ],
  ];
];

```

So the code that “replaces” the `iwant_container` simply does a

```
return $form['iwant_container'];
```

after `buildForm` has run and filled in the `$form['iwant_container']`

Magic!

`ajaxUpdateActionsForm()` cleverly enables the Go button when there is a node stored in the form, otherwise it is disabled (from `buildForm()`). It actually just replaces the whole element with an enabled version:

```

/**
 * Fired from the iwant selection to update the go button.
 */
public function ajaxUpdateActionsForm($form, FormStateInterface $form_state) {
  return $form['actions'];
}

```

This only runs if there is a node, otherwise the Go button is enabled. It seems a little counterintuitive to leave it enabled and only disable if there isn't a node but it would probably work fine the other way around

```

if (!$form_state->get('node')) {
  $form['actions']['submit']['#attributes']['disabled'] = 'disabled';
}

```

Phew! The explanation is much longer than the actual code. Hopefully it is helpful in explaining the intricacies of AJAX Drupal magic.

CUSTOM RESPONSES

Here rather than just returning the part of the form we want to update, we can add a response followed by some JavaScript that should do something. While it seems this should work, sadly it doesn't unless I wrap the function in `(function($) {})(jQuery);`. More below.

```

use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\InvokeCommand;
use Drupal\Core\Ajax\ReplaceCommand;

...

/**
 * Fired from the iam selection to update the iwant select field.
 */
public function ajaxReplaceWantForm($form, FormStateInterface $form_state) {
// return $form['iwant_container'];

$response = new AjaxResponse();
$response->addCommand(new ReplaceCommand('#iwant-container', $form['iwant_container']));
$response->addCommand(new InvokeCommand(NULL, 'initCustomForms'));
return $response;
}

```

I made a new function called myinitCustomForms in app.js (web/themes/custom/tgx/foundation/src/assets/js/app.js).

```

$response->addCommand(new InvokeCommand(NULL, 'myinitCustomForms'));
return $response;

```

And it works a treat!!

At <https://www.drupal.org/docs/8/api/javascript-api/ajax-forms> they suggest an example (which I tried in txg/web/themes/custom/tgx/foundation/src/assets/js/app.js).

```

(function($) {
// Argument passed from InvokeCommand.
$.fn.myAjaxCallback = function(argument) {
  console.log('myAjaxCallback is called.');
  // Set textfield's value to the passed arguments.
  $('#edit-output').attr('value', argument);
};
})(jQuery);

```

This works fine when called with:

```

$response->addCommand(new InvokeCommand(NULL, 'myAjaxCallback', ['This is the new text!']));

```

So my old JavaScript function:

```

// initialize custom form elements
function initCustomForms() {
  jcf.setOptions('Select', {
    maxVisibleItems: 6,
    wrapNative: false,
    wrapNativeOnMobile: false
  });
  jcf.replaceAll();
}

```

Should be wrapped in (function(\$){}

```
(function($) {
  // Argument passed from InvokeCommand.
  $fn.myinitCustomForms = function(argument) {
    console.log('myinitCustomForms is called');
    jcf.setOptions('Select', {
      maxVisibleItems: 6,
      wrapNative: false,
      wrapNativeOnMobile: false
    });
    jcf.replaceAll();
  };
})(jQuery);
```

How do you find all the possible AJAX commands to use with addCommand()?

Just look in docroot/core/lib/Drupal/Core/Ajax

You will see a bunch of different classes - All the Commands you want to use with addCommand() for AJAX response e.g. RedirectCommand or OpenModalDialogCommand.

Another AJAX Submit example

There is another example of a form with an AJAX submit in docroot/modules/custom/quick_pivot/src/Form/QuickPivotSubscribeForm.php.

Note how the callback is explicitly spelled out as:

```
'callback' => 'Drupal\quick_pivot\Form\QuickPivotSubscribeForm::quickPivotAjaxSubmit',
```

Build the form:

```

public function buildForm(array $form, FormStateInterface $form_state) {
  $form['#id'] = 'quick-pivot-subscribe-form';
  $form['#cache'] = ['max-age' => 0];
  $form['#attributes'] = ['autocomplete' => 'off'];

  $form['email'] = [
    '#type' => 'textfield',
    '#id' => 'quick-pivot-email',
    '#placeholder' => $this->t('Email address'),
    '#attributes' => ['class' => ['edit-quick-pivot-email']],
    '#prefix' => '<div class="subscriber-email-msg">',
    '#suffix' => '</div>',
  ];
  $form['actions']['subscribe_submit'] = [
    '#type' => 'submit',
    '#value' => $this->t('Sign Up'),
    '#name' => 'quick_pivot_subscribe_form_submit_button',
    '#ajax' => [
      'callback' => 'Drupal\quick_pivot\Form\QuickPivotSubscribeForm::quickPivotAjaxSubmit',
      'wrapper' => 'quick-pivot-subscribe-form',
      'progress' => ['type' => 'throbber', 'message' => NULL],
    ],
  ];
  $form['message'] = [
    '#type' => 'markup',
    '#markup' => '<div id="quick-pivot-message-area"></div>',
  ];

  return $form;
}

```

Leave the `validateForm()` and `submitForm()` functions empty.

Here is the ajax callback. Note the function has to be static to avoid the possible error:

```
TypeError: Argument 1 passed to Drupal\Core\Routing\RequestContext::fromRequest() must be an instance of Symfony\Component\HttpFoundation\Request, null given
```

This error seems to have something to do with memcache and anonymous users.

```

public static function quickPivotAjaxSubmit(array &$form, FormStateInterface $form_state) {
  $validate = TRUE;
  $email = trim($form_state->getValue('email'));
  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $message = t('Please enter a valid email address.');
    $validate = FALSE;
    $css_border = ['border' => '1px solid red'];
    $css_color = ['color' => 'red'];
  }
  if ($validate) {
    $css_border = ['border' => '1px solid green'];
    $css_color = ['color' => 'green'];
    $response = \Drupal::service('quick_pivot.api')->subscribeEmail($email);
    if (strpos(reset($response), 'Success') !== FALSE) {
      $message = t('Thank you for signing up. Your subscription has been activated.');
    }
    else {
      $message = t('Your subscription could not be processed.');
    }
  }
}

$response = new AjaxResponse();

$quick_pivot_form = \Drupal::formBuilder()->rebuildForm('quick_pivot_subscribe_form', $form_state);
if ($validate) {
  $quick_pivot_form['email']['#value'] = '';
  $quick_pivot_form['email']['#placeholder'] = t('Email address');
}
$response->addCommand(new ReplaceCommand('#quick-pivot-subscribe-form', $quick_pivot_form));
$response->addCommand(new CssCommand('#edit-quick-pivot-email', $css_border));
$response->addCommand(new HtmlCommand('#quick-pivot-message-area', $message));
$response->addCommand(new CssCommand('#quick-pivot-message-area', $css_color));
return $response;
}

```

Config Forms

Generate a config form with drush

Use the [drush command](#) drush generate form:config for a quick boilerplate version.

Config forms overview

Configuration forms extend ConfigFormBase so be sure to

```
use Drupal\Core\Form\ConfigFormBase;
```

The \$config object has get(), set() and save() methods to access config information.

Create a module.settings.yml file in config/install directory for config objects that the module needs.

Also create a module.schema.yml file in config/schema for schema info i.e. definitions and mappings for config objects

Implement the following methods: buildForm(), getEditableConfigNames(), getFormId(), submitForm(), validateForm().

from dev1/web/modules/custom/rsvp/src/Form/RSVPConfigurationForm.php here is the buildForm() function.

```

/**
 * @inheritDoc
 */
public function buildForm(array $form, FormStateInterface $form_state) {
    // $types = node_types_get_names();

    $config = $this->config('rsvp.settings');
    $node_types = \Drupal\node\Entity\NodeType::loadMultiple();
    // If you need to display them in a drop down:
    $options = [];
    foreach ($node_types as $node_type) {
        $options[$node_type->id()] = $node_type->label();
    }

    $form['rsvp_node_types'] = [
        '#type' => 'checkboxes',
        '#title' => $this->t('The content types top enable rSVP collection for'),
        '#default_value' => $config->get('allowed_types'),
        '#options' => $options,
        '#description' => $this->t('On the specified node types, an RSVP option will be available and can be enabled while that node is being edited.'),
    ];
    $form['array_filter'] = ['#type' => 'value', '#value' => TRUE];
    return parent::buildForm($form, $form_state);
}

```

And the submitForm()

```

/**
 * @inheritDoc
 */
public function submitForm(array &$form, FormStateInterface $form_state) {
    $allowed_types = array_filter($form_state->getValue('rsvp_node_types'));
    sort($allowed_types);
    $this->config('rsvp.settings')
        ->set('allowed_types', $allowed_types)
        ->save();

    parent::submitForm($form, $form_state);
}

```

To set a default value for when the module is first installed, create web/modules/custom/rsvp/config/install/rsvp.settings.yml

```

allowed_types:
  - article

```

and to specify more details of what the config stores, create the /Users/selwyn/Sites/dev1/web/modules/custom/rsvp/config/schema/rsvp.schema.yml.

```

#Schema for config file of rsvp module
rsvp.admin_settings:
  type: config_object
  label: 'RSVP Content Type Settings'
  mapping:
    allowed_types:
      type: sequence
      label: 'Content types RSVP form can display on'
      sequence:
        type: string
        label: 'Content type'

```

The basics of implementing forms

Location

Forms are stored in `modules/src/Form/MyClassForm.php` e.g. `/modules/custom/dmod/src/Form/HeaderFooterForm.php`.

Base Classes for forms

- `FormBase` – for any old form
- `ConfirmFormBase` – for generic confirmation form
- `ConfigFormBase` – for config forms

Create your form class by extending `Formbase`

```
class HeaderFooterForm extends FormBase {
```

The main methods

Forms typically need a `buildForm()`, `submitForm()` and `getFormId()` member function. Validation is handled with a `validateForm()` member function. Each is explained with more detail below:

GETFORMID()

It only returns a simple string identifying the form, e.g.

```
public function getFormId() {
    return 'dan_header_footer_form';
}
```

BUILDFORM()

Create the render array representing the form elements. Here I am using a fieldset to group fields. Also the `'#default_value'` provides a default value that users can edit.

```

$form['instructions'] = [
  '#type' => 'markup',
  '#markup' => $this->t('After clicking submit, test changes on the header and footer of the home page.<br/><br/>')
];
$form['header'] = [
  '#type' => 'fieldset',
  '#title' => $this->t('Header'),
  '#collapsible' => TRUE,
  '#open' => TRUE,
];
$form['header'][\logo_url] = [
  '#type' => 'url',
  '#title' => $this->t('Logo URL'),
  '#size' => 64,
  '#default_value' => $logo_url,
];
$form['footer'] = [
  '#type' => 'fieldset',
  '#title' => $this->t('Footer'),
  '#collapsible' => TRUE,
  '#open' => TRUE,
];
$form['footer'][\footer_address1] = [
  '#type' => 'textfield',
  '#title' => $this->t('Address line 1'),
  '#default_value' => $address1,
];
// You need a submit button so users can click on something.
$form['submit'] = [
  '#type' => 'submit',
  '#value' => $this->t('Save'),
];

```

Don't **EVER EVER** forget to return the \$form render array you just created otherwise you get an empty form. Not that it ever ever happened to me ;-)

```
return $form;
```

SUBMITFORM()

When a user clicks on the submit button, this function is called. You can extract the entries from the \$form_state and decide what to do with them. In this example we write them into the database using the State API. State API values are stored in the key_value table.

```

$values = $form_state->getValues();

$address1 = $values[\footer_address1];
$address2 = $values[\footer_address2];
$address3 = $values[\footer_address3];
$email = $values[\email];
$logo_url = $values[\logo_url];
$facebook = $values[\facebook];
$linkedin = $values[\linkedin];
$instagram = $values[\instagram];
$twitter = $values[\twitter];
$youtube = $values[\youtube];

```

Writing:

```
\Drupal::state()->set('logo_url', $logo_url);
\Drupal::state()->set('footer_address1', $address1);
\Drupal::state()->set('footer_address2', $address2);
\Drupal::state()->set('footer_address3', $address3);
\Drupal::state()->set('footer_email', $email);
\Drupal::state()->set('footer_facebook', $facebook);
\Drupal::state()->set('footer_instagram', $instagram);
\Drupal::state()->set('footer_linkedin', $linkedin);
\Drupal::state()->set('footer_twitter', $twitter);
\Drupal::state()->set('footer_youtube', $youtube);

\Drupal::messenger()->addMessage('Values updated');
```

To display the results of the form entry you can put this in submitForm().

```
// Display result.
foreach ($form_state->getValues() as $key => $value) {
  \Drupal::messenger()
    ->addMessage($key . ': ' . ($key === 'text_format' ? $value['value'] : $value));
}
```

Form validation example #1

Check a string length for the company_name field.

```
public function validateForm(array &$form, FormStateInterface $formState) {
  if (!($formState->isValueEmpty('company_name'))) {
    if (strlen($formState->getValue('company_name')) <= 5) {
      //Set validation error.
      $formState->setErrorByName('company_name', t('Company name is less than 5 characters'));
    }
  }
}
```

Form Validation example #2

From web/modules/custom/rsvp/src/Form/RSVPForm.php we call Drupal's email.validator service and if it fails, setErrorByName()

```
public function validateForm(array &$form, FormStateInterface $form_state) {
  $value = $form_state->getValue('email');
  if (!\Drupal::service('email.validator')->isValid($value)) {
    $form_state->setErrorByName('email', t('The email %mail is not valid.', ['%mail'=> $value]));
  }

  parent::validateForm($form, $form_state);
}
```

Field attributes

They always begin with #. You can find all the possible attributes at [Form Element Reference with examples](#)

```

$form['email'] = [
  '#title' => t('Email Address'),
  '#type' => 'textfield',
  '#size' => 25,
  '#description' => t("We'll send updates to the email address"),
  '#required' => TRUE,
];
$form['submit'] = [
  '#type' => 'submit',
  '#value' => t('RSVP'),
];
$form['nid'] = [
  '#type' => 'hidden',
  '#value' => $nid,
];

```

Form Elements

[Form Element Reference with examples](#)

You can add markup to a form e.g. at web/modules/custom/modal_form_example/src/Form/ExampleForm.php

```

public function buildForm(array $form, FormStateInterface $form_state, $options = NULL) {

  $form['zzz'] = [
    '#type' => 'markup',
    '#markup' => $this->t('this is a test'),
  ];

  $form['name'] = [
    '#type' => 'textfield',
    '#title' => $this->t('Name'),
    '#size' => 20,
    '#default_value' => 'Joe Blow',
    '#required' => FALSE,
  ];
}

```

and prefixes and suffices

```

public function buildForm(array $form, FormStateInterface $form_state, $options = NULL) {

  $form['#prefix'] = '<div id="example_form">';
  $form['#suffix'] = '</div>';
}

```

Hidden or required via a custom class

```

'#maxlength' => 40,
'#attributes' => [
  'class' => ['hidden'],
],

```

Another example:

```
'first_name' => [
  '#type' => 'textfield',
  '#title' => 'First Name',
  '#default_value' => '',
  '#size' => 25,
  '#maxlength' => 40,
  '#attributes' => [
    'class' => ['required name-on-card'],
  ],
  '#field_prefix' => '<span class="error-msg">*</span>',
],
```

Retrieving field values

When you go to grab a field value from a form, use `getValue()`. The values that come back from this are arrays so you have to extract them like this (from `org_mods.module`)

```
function cn_form_validate($form, FormStateInterface $form_state) {
  $extension = $form_state->getValue('field_cn_extension');
  if (is_array($extension)) {
    $extension = $extension['value'];
  }
}
```

Alternatively, you can get all the fields at one time with `getValues()`. And reference their value like this.

```
$values = $form_state->getValues();

if ($values['op'] == 'Goback') {
  ...
}
```

If you define the form with fields like `$form['header']['blah'] = ...` then you can retrieve those with `$form_state->getValue('header', 'blah');`

or

if you define a checkbox like this:

```
$form['actions']['delete_extras'] = [
  '#type' => 'checkbox',
  '#title' => t('Also delete extra items'),
  '#required' => FALSE,
  '#default_value' => FALSE,
  '#description' => $this->t('Checking this box will delete extra items also.'),
```

You can retrieve the value of the checkbox with:

```
$delete_extras = $form_state->getValue('delete_extras');

// And then use it.

if (!$delete_extras) {
  $query->condition('field_extras', '', '<>');
}
```

Resources

- [Drupal API Form Element Reference with examples](#)
- [Drupal AJAX AJAX forms updated Dec 2022](#)
- [Drupal AJAX Dialog boxes updated Nov 2022](#)
- [Great tutorial from Mediurrent on using modal forms in Drupal from Mar 2017](#)
- [Form API Internal Workflow updated Dec 2022](#)

- [#! code: Drupal 9: Creating A GET Form, July 2021](#)
 - [Conditional fields module](#)
-
-

[Back to top](#)

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

Page last modified: Jun 21 2023.

[Edit this page on GitHub](#)

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

General

TABLE OF CONTENTS

- [Get the current user](#)
- [Get the logged in user name and email](#)
- [Check if you are on the Front page](#)
- [Check if site is in system maintenance mode](#)
- [Get Node URL alias or Taxonomy Alias by Node id or Term ID](#)
- [Taxonomy alias](#)
- [Get current Path](#)
- [Get current nid, node type and title](#)
- [How to check whether a module is installed or not](#)
- [Get current Route name](#)
- [Get the current page title](#)
- [Get the current user](#)
- [Check if you are on the Front page](#)
- [Check if site in system maintenance mode](#)
- [Retrieve query and get or post parameters \(\\$_POST and \\$_GET\)](#)
- [Retrieve URL argument parameters](#)
- [Get Current Language in a constructor](#)
- [Add a variable to any page on the site](#)
- [Add a variable to be rendered in a node.](#)
- [Add a bunch of variables to be rendered in a node](#)
- [Grabbing entity reference fields in hook_preprocess_node for injection into the twig template](#)
- [Render a list created in the template_preprocess_node\(\)](#)
- [Indexing paragraphs so you can theme the first one](#)
- [Add meta tags using template_preprocess_html](#)
- [How to strip % characters from a string](#)
- [Remote media entities](#)
- [Deprecated functions like drupal_set_message](#)
- [Block excessive crawling of Drupal Views or search results](#)

views 107

Get the current user

Note this will not get the user entity, but rather a user proxy with basic info but no fields or entity-specific data.

```
$user = \Drupal::currentUser();
```

```
$user = \Drupal\user\Entity\User::load(\Drupal::currentUser()->id());
```

Or

```
use \Drupal\user\Entity\User;
$user = User::load(\Drupal::currentUser()->id());
```

Get the logged in user name and email

```
$username = \Drupal::currentUser()->getAccountName();
```

or

```
$account_proxy = \Drupal::currentUser();
// $account = $account_proxy->getAccount();
```

```
// load user entity
$user = User::load($account_proxy->id());
$user = User::load(\Drupal::currentUser()->id());
$name = $user->get('name')->value;
```

Email

```
$email = \Drupal::currentUser()->getEmail();
```

or

```
$user = User::load(\Drupal::currentUser()->id());
$email = $user->get('mail')->value;
```

Check if you are on the Front page

```
$is_front = \Drupal::service('path.matcher')->isFrontPage();
```

The above statement will return either TRUE or FALSE. TRUE means you are on the front page.

Check if site is in system maintenance mode

```
$is_maint_mode = \Drupal::state()->get('system.maintenance_mode');
```

Get Node URL alias or Taxonomy Alias by Node id or Term ID

Sometimes we need a relative path and sometimes we need an absolute path. There is an \$options parameter in the fromRoute() function where specify which you need.

Parameters:

- absolute true will return absolute path.
- absolute false will return relative path.

Returns the node alias. Note. If a nice url is not set using pathauto, you get /node/1234

```

use Drupal\Core\Url;
$options = ['absolute' => true]; //false will return relative path.

$url = Url::fromRoute('entity.node.canonical', ['node' => 1234], $options);
$url = $url->toString(); // make a string

// OR

$node_path = "/node/1";
$alias = \Drupal::service('path.alias.manager')->getAliasByPath($node_path);

// OR

$current_path = \Drupal::service('path.current')->getPath();

```

To get the full path with the host etc. this returns: <https://ddev93.ddev.site/node/1>

```

$host = \Drupal::request()->getSchemeAndHttpHost();
$url = \Drupal\Core\Url::fromRoute('entity.node.canonical',['node'=>$lab_home_nid]);
$url_alias = $url->toString();
$full_url = $host . $url->toString();

```

You can get the hostname, e.g. "drupal8.local", directly from the `getHost()` request with:

```
$host = \Drupal::request()->getHost();
```

Taxonomy alias

Return taxonomy alias

```

$options = ['absolute' => true]; //false will return relative path.
$url = Url::fromRoute('entity.taxonomy_term.canonical', ['taxonomy_term' => 1234], $options);

```

Get current Path

For node pages this will return `node/{node id}`, for taxonomy `taxonomy/term/{term id}`, for user `user/{user id}` if exists otherwise it will return the current request URI.

```
$currentPath = \Drupal::service('path.current')->getPath();
```

Get current nid, node type and title

There are two ways to retrieve the current node – via the request or the route

```

$node = \Drupal::request()->attributes->get('node');
$nid = $node->id();

```

OR

```

$node = \Drupal::routeMatch()->getParameter('node');
if ($node instanceof \Drupal\node\NodeInterface) {
  // You can get nid and anything else you need from the node object.
  $nid = $node->id();
  $nodeType = $node->bundle();
  $nodeTitle = $node->getTitle();
}

```

If you need to use the `node` object in `hook_preprocess_page` on the preview page, you will need to use the `node_preview` parameter, instead of the

node parameter:

```
function mymodule_preprocess_page(&$vars) {  
  
  $route_name = \Drupal::routeMatch()->getRouteName();  
  
  if ($route_name == 'entity.node.canonical') {  
    $node = \Drupal::routeMatch()->getParameter('node');  
  }  
  elseif ($route_name == 'entity.node.preview') {  
    $node = \Drupal::routeMatch()->getParameter('node_preview');  
  }  
}
```

And from <https://drupal.stackexchange.com/questions/145823/how-do-i-get-the-current-node-id> when you are using or creating a custom block then you have to follow this code to get current node id. Not sure if it is correct.

```
use Drupal\Core\Cache\Cache;  
  
$node = \Drupal::routeMatch()->getParameter('node');  
if ($node instanceof \Drupal\node\NodeInterface) {  
  $nid = $node->id();  
}  
  
// for cache  
public function getCacheTags() {  
  //With this when your node changes your block will rebuild  
  if ($node = \Drupal::routeMatch()->getParameter('node')) {  
    //if there is node add its cachetag  
    return Cache::mergeTags(parent::getCacheTags(), ['node' . $node->id()]);  
  }  
  else {  
    //Return default tags instead.  
    return parent::getCacheTags();  
  }  
}  
  
public function getCacheContexts() {  
  //if you depend on \Drupal::routeMatch()  
  //you must set context of this block with 'route' context tag.  
  //Every new route this block will rebuild  
  return Cache::mergeContexts(parent::getCacheContexts(), ['route']);  
}
```

How to check whether a module is installed or not

```
$moduleHandler = \Drupal::service('module_handler');  
$module_name = 'views';  
if ($moduleHandler->moduleExists($module_name)) {  
  echo "$module_name installed";  
}  
else {  
  echo "$module_name not installed";  
}
```

Get current Route name

Routes are in the form: view.files_browser.page_1, test.example or test.settings_form

E.g from test.routing.yml

```

test.example:
path: '/test/example'
defaults:
  _title: 'Example'
  _controller: '\Drupal\test\Controller\TestController::build'
requirements:
  _permission: 'access content'

test.settings_form:
path: '/admin/config/system/test'
defaults:
  _title: 'Test settings'
  _form: 'Drupal\test\Form\SettingsForm'
requirements:
  _permission: 'administer test configuration'

```

This will return Drupal route. It returns entity.node.canonical for the nodes, system.404 for the 404 pages, entity.taxonomy_term.canonical for the taxonomy pages, entity.user.canonical for the users and custom route name that we define in modulename.routing.yml file.

```
$current_route = \Drupal::routeMatch()->getRouteName();
```

Get the current page title

You can use this in a controller, to return the current page title.

```

$request = \Drupal::request();
if ($route = $request->attributes->get('Symfony\Cmf\Component\Routing\RouteObjectInterface::ROUTE_OBJECT')) {
$title = \Drupal::service('title_resolver')->getTitle($request, $route);}

```

Get the current user

Note this will not get the user entity, but rather a user proxy with basic info but no fields or entity-specific data.

```
$user = \Drupal::currentUser();
```

To get the user entity, use this which gets the user service (\Drupal::currentUser()), gets the uid (->id()), then calls load() to load the real user object.

```
$user = \Drupal\user\Entity\User::load(\Drupal::currentUser()->id());
```

Or

```
use \Drupal\user\Entity\User;
$user = User::load(\Drupal::currentUser()->id());
```

Check if you are on the Front page

This will return true for the front page otherwise false.

```
$is_front = \Drupal::service('path.matcher')->isFrontPage();
```

Check if site in system maintenance mode

From web/core/modules/system/src/Access/CronAccessCheck.php

```
if (\Drupal::state()->get('system.maintenance_mode')) {
```

Retrieve query and get or post parameters (\$_POST and \$_GET)

Old style was:

```
$name = $_POST['name'];
```

Now use this for post vars:

```
$name = \Drupal::request()->request->get('name');
```

And this for gets

```
$query = \Drupal::request()->query->get('name');
```

For all items in get:

```
$query = \Drupal::request()->query->all();
$search_term = $query['query'];
$collection = $query['collection'];
```

Be wary about caching. From <https://drupal.stackexchange.com/questions/231953/get-in-drupal-8/231954#231954> the code provided only works the first time so it is important to add a '#cache' context in the markup.

```
namespace Drupal\newday\Controller;
use Drupal\Core\Controller\ControllerBase;

class NewdayController extends ControllerBase {
  public function new() {
    $day = [
      "#markup" => \Drupal::request()->query->get('id'),
    ];
    return $day;
  }
}
```

The request is being cached, you need to tell the system to vary by the query arg:

```
$day = [
  '#markup' => \Drupal::request()->query->get('id'),
  '#cache' => [
    'contexts' => ['url.query_args:id'],
  ],
];
```

More about caching render arrays: <https://www.drupal.org/docs/8/api/render-api/cacheability-of-render-arrays>

Retrieve URL argument parameters

You can extract the url arguments with

```
$current_path = \Drupal::service('path.current')->getPath();
$path_args = explode('/', $current_path);
$term_name = $path_args[3];
```

For <https://txg.ddev.site/newsroom/search/?country=1206>

```

01 $current_path = "/newsroom/search"
▼ 1 $path_args = {array} [3]
  0 = ""
  1 = "newsroom"
  2 = "search"

```

Get Current Language in a constructor

In dev1 - /modules/custom/iai_wea/src/Plugin/rest/resource/WEAResource.php we create the WeaResource class and using dependency injection, get the LanguageManagerInterface service passed in, then we call getCurrentLanguage(). This allows us to later retrieve the node

```

class WEAResource extends ResourceBase {

    /**
     * @var \Drupal\Core\Language\Language
     */
    protected $currentLanguage;

    /**
     * WEAResource constructor.
     *
     * @param array $configuration
     * @param string $plugin_id
     * @param mixed $plugin_definition
     * @param array $serializer_formats
     * @param \Psr\Log\LoggerInterface $logger
     * @param \Drupal\Core\Language\LanguageManagerInterface $language_manager
     */
    public function __construct(array $configuration, string $plugin_id, mixed $plugin_definition, array $serializer_formats, \Psr\Log\LoggerInterface $logger, LanguageManagerInterface $language_manager) {
        parent::__construct($configuration, $plugin_id, $plugin_definition, $serializer_formats, $logger);
        $this->currentLanguage = $language_manager->getCurrentLanguage();
    }
}

```

Later in the class, we can retrieve the correct language version of the node:

```

public function get($id) {
    if ($node = Node::load($id)) {
        $translatedNode = $node->getTranslation($this->currentLanguage->getId());
    }
}

```

Of course, you can also get the language statically by using:

```
Global $language = Drupal::languageManager()->getLanguage(Language::TYPE_INTERFACE)
```

This is part of the packt publishing Mastering Drupal 8 module development video series: <https://www.packtpub.com/product/mastering-drupal-8-development-video/9781787124493>

Note. To test this in modules/custom/pseudo_client/get/

```
php -S localhost:8888
```

and put this in a browser:

```
http://localhost:8888/get_item_from_drupal_core.php?domain=dev1&item=2716
```

or

http://localhost:8888/get_items_from_custom_code.php?domain=dev1

OR just put this in browser without running php -S:

http://dev1/iai_wea/actions/2716?_format=json

Add a variable to any page on the site

In the .theme file of the theme, add a hook preprocess_page function like in themes/custom/dprime/dprime.theme:

```
function dprime_preprocess_page(&$variables) {
  $language_interface = \Drupal::languageManager()->getCurrentLanguage();

  $variables['footer_address1'] = [
    '#type'=>'markup',
    '#markup'=>'123 Disk Drive, Sector 439',
  ];
  $variables['footer_address2'] = [
    '#type'=>'markup',
    '#markup'=>'Austin, Texas 78759',
  ];
}
```

Then in the template file e.g. themes/custom/dprime/templates/partials/footer.html.twig

```
<div class="cell xlarge-3 medium-4">
<address>
<br />
<br />
  Campus mail code: D9000<br />
  <a href="mailto:abc@example.com">abc@example.com </a>
</address>
</div>
```

Add a variable to be rendered in a node.

From dev1 custom theme burger_burgler.

Here two vars stock_field and my_custom_field are added and will be rendered by a normal node twig file. The function hook preprocess_node is in the .theme file at themes/custom/burger_burgler/burger_burgler.theme.

```
function burger_burgler_preprocess_node(&$variables) {

  $variables['content']['stock_field'] = [
    '#type'=>'markup',
    '#markup'=>'stock field here',
  ];

  $variables['content']['my_custom_field'] = [
    '#type' => 'markup',
    '#markup' => 'Hello - custom field here',
  ];
}
```

If you've tweaked your node twig template, you'll need to reference like this:

```
<div class="stock-field-class">
{{ content['stock_field'] }}
</div>
```

Note. You can always just add a variable like

```
$variables['abc'] = 'hello';
```

which can be referenced in the template as {{ abc }} (or {{ kint(abc) }})

Add a bunch of variables to be rendered in a node

You can easily grab the node from the \$variables with:

```
$node = $variables['node'];
```

Then to access a field in the node, you can just specify them by:

```
$node->field_ref_aof  
$node->field_ref_topic
```

Here we grab a bunch of variables, cycles through them (for multi-value fields, which most of them are and build an array that can be easily rendered by twig:

From: themes/custom/txg/txg.theme

```

function txg_preprocess_node(&$variables) {
  $view_mode = $variables['view_mode']; // Retrieve view mode
  $allowed_view_modes = ['full']; // Array of allowed view modes (for performance so as to not execute on unneeded nodes)
  $node = $variables['node'];

  if (($node->getType() == 'news_story') && ($view_mode == 'full')) {
    $aofs = _txg_multival_ref_data($node->field_ref_aof, 'aof', 'target_id');
    $units = _txg_multival_ref_data($node->field_ref_unit, 'unit', 'target_id');
    $audiences = _txg_multival_ref_data($node->field_ref_audience, 'audience', 'target_id');
    $collections = _txg_multival_ref_data($node->field_ref_program_collection, 'collection', 'target_id');
    $topics = _txg_multival_ref_data($node->field_ref_topic, 'topic', 'target_id', 'taxonomy');
    $continents = _txg_multival_ref_data($node->field_continent, 'continent', 'value', 'list');
    $countries = _txg_multival_ref_data($node->field_ref_country, 'country', 'target_id');

    $related_news_items = array_merge($topics, $aofs, $units, $audiences, $collections, $continents, $countries);
    $variables['related_news_items'] = $related_news_items;
  }
}

/**
 * Returns array of data for multivalue node reference fields
 * ref_field = entity reference field
 * param_name = parameter name to be passed as get value
 * value_type = indicates which field to retrieve from database
 * field_ref_type = variable to determine type of reference field
 * field_term_category =
 */
function _txg_multival_ref_data($ref_field, $param_name, $value_type, $field_ref_type = 'node') {
  $values = [];
  foreach($ref_field as $ref) {
    if ($field_ref_type == 'taxonomy') {
      $term = Drupal::entityTypeManager()->getStorage('taxonomy_term')->load($ref->$value_type);
      $title = $term->getName();
    }
    else {
      $title = $value_type == 'value' ? $ref->$value_type : $ref->entity->title->value;
    }
    $id = $ref->$value_type;
    $values[] = [
      'title' => $title,
      'id' => str_replace(' ', '+', $id),
      'param_name' => $param_name,
    ];
  }
  return $values;
}

```

Grabbing entity reference fields in hook_preprocess_node for injection into the twig template

You can easily pull in referenced fields by referring to them as

```
$node->field_sf_contract_ref->entity->field_how_to_order->value;
```

Where `field_sf_contract_ref` is the reference field, which points to an entity which has a field called `field_how_to_order`. Then we can jam it into the `$variables` array and refer to it in the twig template as ``

From `web/themes/custom/dirt_bootstrap/dirt_bootstrap.theme`

In function `dirt_bootstrap_preprocess_node(&$variables)`

```

if ($type === 'contract') {
  if ($view_mode === 'full') {
    $how_to_order_lookup = $node->field_sf_contract_ref->entity->field_how_to_order_lookup->value;
    $variables['how_to_order_lookup'] = $how_to_order_lookup;
    $contract_type = $node->get('field_contract_type')->value;
    if ($how_to_order_lookup === "Custom Text") {
      if ($contract_type === "DIRT") {
        $variables['how_to_order'] = $node->field_sf_contract_ref->entity->field_how_to_order->value;
      }
      else {
        $variables['how_to_order'] = $node->field_sf_contract_ref->entity->field_how_to_order_custom->value;
      }
    }
  }
}

```

Render a list created in the template_preprocess_node()

Here we create a list in the preprocess_node custom theme burger_burgler):

```

function burger_burgler_preprocess_node(&$variables) {

  $burger_list = [
    ['name' => 'Cheesburger'],
    ['name' => 'Mushroom Swissburger'],
    ['name' => 'Jalapeno bugburger'],
  ];
  $variables['burgers'] = $burger_list;

}

```

and render it in the twig template node--article--full.html.twig

```

<ol>
  {% for burger in burgers %}
    <li> {{ burger['name'] }} </li>
  {% endfor %}
</ol>

```

Indexing paragraphs so you can theme the first one

Posted on <https://www.drupal.org/project/paragraphs/issues/2881460#comment-13291215>

From themes/custom/dprime/dprime.theme

Add this to the theme:

```

/**
 * Implements hook_preprocess_field.
 *
 * Provides an index for these fields referenced as
 * in twig template.
 *
 * @param $variables
 */
function dprime_preprocess_field(&$variables) {
  if($variables['field_name'] == 'field_video_accordions'){
    foreach($variables['items'] as $idx => $item) {
      $variables['items'][$idx]['content'][">#paragraph']->index = $idx;
    }
  }
}

```

`field_video_accordions` is the name of the field that holds the paragraph you want to count.

In the twig template for that paragraph, you can use the value `paragraph.index` as in:

```

{% if paragraph.index == 0 %}
<li class="accordion-item is-active" data-accordion-item="">
{% else %}
<li class="accordion-item" data-accordion-item="">
{% endif %}

```

Add meta tags using template_preprocess_html

Also covered at <https://drupal.stackexchange.com/questions/217880/how-do-i-add-a-meta-tag-in-inside-the-head-tag>

If you need to make changes to the `<head>` element, the `hook_preprocess_html` is the place to do it in the `.theme` file. Here we check to see that the content type is contract and then we create a fake array of meta tags and jam them into the `$variables['page'][#attached]['html_head']` element. They are then rendered on the page.

```

/**
 * Implements hook_preprocess_html().
 */
function dir_bootstrap_preprocess_html(&$variables) {

$node = \Drupal::routeMatch()->getParameter('node');
if ($node instanceof \Drupal\node\NodeInterface) {
  if ($node->getType() == 'contract') {

$brand_meta_tag[] = [
  '#tag' => 'meta',
  '#attributes' => [
    'name' => 'brand',
    'content' => 'Dell',
  ],
];
$brand_meta_tag[] = [
  '#tag' => 'meta',
  '#attributes' => [
    'name' => 'description',
    'content' => 'Dell',
  ],
];
$brand_meta_tag[] = [
  '#tag' => 'meta',
  '#attributes' => [
    'name' => 'keywords',
    'content' => 'Dell',
  ],
];

$variables['page'][#attached]['html_head'][] = $brand_meta_tag;
}
}

```

Note that the extra "Dell" low down in the array appears to be a description of some kind – it isn't rendered. If you don't include the second "Dell" you could rather use

```
$page['#attached']['html_head'][] = [$description, 'description'];
```

For multiple tags, I had to do this version:

```
$brand_meta_tags[] = [[
  '#tag' => 'meta',
  '#attributes' => [
    'name' => 'brand',
    'content' => 'Dell',
  ],
  'Dell',
];
$brand_meta_tags[] = [[
  '#tag' => 'meta',
  '#attributes' => [
    'name' => 'brand',
    'content' => 'Apple',
  ],
  'Apple',
];
foreach ($brand_meta_tags as $brand_meta_tag) {
  $variables['page']['#attached']['html_head'][] = $brand_meta_tag;
}
```

And here I do a query and build some new meta tags from themes/custom/dirt_bootstrap/dirt_bootstrap.theme.

```
$brand_meta_tags = [];
$contract_id = $node->field_contract_id->value;
if ($contract_id) {

  //Lookup dirt store brand records with this contract id.
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'sf_store_brands')
    ->condition('status', 1)
    ->condition('field_contract_id', $contract_id);
  $nids = $query->execute();
  foreach ($nids as $nid) {
    $store_brand_node = Node::load($nid);
    $brand = $store_brand_node->field_brand->value;
    if ($brand) {
      $brand_meta_tags[] = [[
        '#tag' => 'meta',
        '#attributes' => [
          'name' => 'brand',
          'content' => $brand,
        ],
        $brand,
      ];
    }
  }
  foreach ($brand_meta_tags as $brand_meta_tag) {
    $variables['page']['#attached']['html_head'][] = $brand_meta_tag;
  }
}
```

How to strip % characters from a string

```

$str = "threatgeek/2016/05/welcome-jungle-tips-staying-secure-when-you're-road";
echo $str . "\n";
//echo htmlspecialchars_decode($str) . "\n";

echo (urldecode($str)) . "\n";
echo urlencode("threatgeek/2016/05/welcome-jungle-tips-staying-secure-when-you're-road");

echo urldecode('We%27re%20proud%20to%20introduce%20the%20Amazing');

```

Remote media entities

For this project, I had to figure out a way to make media entities that really were remote images. i.e. the API provided images but we didn't want to store them in Drupal

I started by looking at <https://www.drupal.org/sandbox/nickhope/3001154> which was based on https://www.drupal.org/project/media_entity_flickr.

I tweaked the nickhope module (media_entity_remote_file) so it worked but it had some trouble with image styles and thumbnails

A good solution (thanks to Hugo) is:

https://www.drupal.org/project/remote_stream_wrapper_widget

https://www.drupal.org/project/remote_stream_wrapper

Hugo suggests using this to do migration:

```

$uri = 'http://example.com/somefile.mp3';
$file = File::Create(['uri' => $uri]);
$file->save();
$node->field_file->setValue(['target_id' => $file->id()]);
$node->save();

```

There was no documentation so I added some at https://www.drupal.org/project/remote_stream_wrapper/issues/2875444#comment-12881516

Deprecated functions like drupal_set_message

Note. drupal_set_message() has been removed from the codebase so you should use messenger() but you can also use dsm() which is provided by the [devel](#) contrib module. This is useful when working through a problem if you want to display a message on a site during debugging.

From <https://github.com/mglaman/drupal-check/wiki/Deprecation-Error-Solutions>

Before

```
drupal_set_message($message, $type, $repeat);
```

After

```
\Drupal::messenger()->addMessage($message, $type, $repeat);
```

[Read more.](#)

Block excessive crawling of Drupal Views or search results

[From Acquia.com](#)

Sometimes, robot webcrawlers (like Bing, Huawei Cloud, Yandex, Semrush, etc) can attempt to crawl a Drupal View's search results pages, and could also be following links to each of the view's filtering options. This places extra load on your site. Additionally, the crawling (even if done by legitimate search engines) may not be increasing your site's visibility to users of search engines.

Therefore, we suggest blocking or re-routing this traffic to reduce resource consumption at the Acquia platform, avoid overages to your

Acquia entitlements (for Acquia Search, Views & Visits, etc.), and to generally help your site perform better.

```
# EXAMPLE ROBOT BLOCKING CODE for Search pages or views.  
# From: https://support-acquia.force.com/s/article/4408794498199-Block-excessive-crawling-of-Drupal-Views-or-search-results  
# NOTE: May need editing depending on your use case(s).  
#  
# INSTRUCTIONS:  
# PLACE THIS BLOCK directly after the "RewriteEngine on" line  
# on your docroot/.htaccess file.  
#  
# This will block some known robots/crawlers on URLs when query arguments are present.  
# DOES allow basic URLs like /news/feed, /node/1 or /rss, etc.  
# BLOCKS only when search arguments are present like  
# /news/feed?search=XXX or /rss?page=21.  
# Note: You can add more conditions if needed.  
# For example, to only block on URLs that begin with '/search', add this  
# line before the RewriteRule:  
# RewriteCond %{REQUEST_URI} ^/search  
#  
RewriteCond %{QUERY_STRING} .  
RewriteCond %{HTTP_USER_AGENT} "11A465|AddThis.com|AdsBot-Google|Ahrefs|alexa site audit|Amazonbot|Amazon-Route53-Health-Check-Service|ApacheBench|AppDy  
RewriteRule ^.* - [F,L]
```

[Back to top](#)

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

Page last modified: Jul 31 2023.

[Edit this page on GitHub](#)

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

Getting off the Island (formerly Reaching out of Drupal)

TABLE OF CONTENTS

- [Overview](#)
- [Guzzle example](#)
- [Guzzle POST example](#)
- [Magic methods to send synchronous requests](#)
- [HTTP basic authentication](#)
- [Exception handling](#)
- [Guzzle Exceptions](#)
- [HTTP response status codes](#)
- [Reading from an API](#)
- [Download a file using guzzle](#)
- [Download a file using curl in PHP](#)
- [Resources](#)

views 144

Overview

To communicate with external websites or web services we can make web requests via the [Drupal:httpClient](#) class. This is a wrapper for the [Guzzle HTTP Client](#).

From <https://www.php-fig.org/psr/psr-7/> HTTP messages are the foundation of web development. Web browsers and HTTP clients such as CURL create HTTP request messages that are sent to a web server, which provides an HTTP response message. Server-side code receives an HTTP request message, and returns an HTTP response message.

HTTP messages are typically abstracted from the end-user consumer, but as developers, we typically need to know how they are structured and how to access or manipulate them in order to perform our tasks, whether that might be making a request to an HTTP API, or handling an incoming request.

Guzzle utilizes [PSR-7](#) as the HTTP message interface. [PSR-7](#) describes common interfaces for representing HTTP messages. This allows Guzzle to work with any other library that utilizes PSR-7 message interfaces.

Guzzle example

This is an example which retrieves data from within a controller using a GET:

```

public function example1() {

    //Initialize client;
    $client = \Drupal::httpClient();
    $uri = 'https://demo.ckan.org/api/3/action/package_list';

    // Returns a GuzzleHttp\Psr7\Response.
    $response = $client->request('GET', 'https://demo.ckan.org/api/3/action/package_list');
    // Or using the magic method.

    $response = $client->get($uri);

    // Returns a GuzzleHttp\Psr7\Stream.
    $stream = $response->getBody();
    $json_data = Json::decode($stream);

    $help = $json_data['help'];
    $success = $json_data['success'];
    $result = $json_data['result'][0];

    $msg = "<br>URI: " . $uri;
    $msg .= "<br>Help: " . $help;
    $msg .= "<br>Success: " . $success;
    $msg .= "<br>Result: " . $result;

    $build['content'] = [
        '#type' => 'item',
        '#markup' => $this->t($msg),
    ];
}

return $build;
}

```

Guzzle POST example

```

//Initialize client;
$client = \Drupal::httpClient();
$uri = 'http://demo.ckan.org/api/3/action/group_list';
$request = $client->post($uri, [
    'json' => [
        'id'=> 'data-explorer'
    ]
]);
$stream = $request->getBody();
$json_data = Json::decode($stream);
$help = $json_data['help'];
$success = $json_data['success'];
$result = $json_data['result'][0] . ' and ' . $json_data['result'][1];

```

Guzzle takes care of adding a 'Content-Type','application/json' header, as well as json_encoding the 'json' array.

Magic methods to send synchronous requests

```
$response = $client->get('http://httpbin.org/get');
$response = $client->delete('http://httpbin.org/delete');
$response = $client->head('http://httpbin.org/get');
$response = $client->options('http://httpbin.org/get');
$response = $client->patch('http://httpbin.org/patch');
$response = $client->post('http://httpbin.org/post');
$response = $client->put('http://httpbin.org/put');
```

From <https://docs.guzzlephp.org/en/latest/quickstart.html#sending-requests>

HTTP basic authentication

This shows a failed attempt to authenticate with Github's API with exception handling. It will log the error to Drupal's watchdog and display it on screen:

```
public function example2() {

$msg = '';
$client = \Drupal::httpClient();
$uri = 'https://api.github.com/user';

try {
$request = $client->get($uri, [
'auth' => ['username', 'password']
]);
$response = $request->getBody();
$msg .= "<br><strong>GET</strong>" ;
$msg .= "<br>URI: " . $uri;
}

catch (ClientException $e) {
\Drupal::messenger()->addError($e->getMessage());
watchdog_exception('guzzle_examples', $e);
}

catch (\Exception $e) {
\Drupal::messenger()->addError($e->getMessage());
watchdog_exception('guzzle_examples', $e);
}

$build['content'] = [
'#type' => 'item',
'#markup' => $this->t($msg),
];
}

return $build;
}
```

See <https://docs.guzzlephp.org/en/latest/request-options.html#auth> for more.

Exception handling

When using `Drupal::httpClient`, you should always wrap your requests in a try/catch block, to handle any exceptions. Here is an example of logging `Drupal::httpClient` request exceptions via `watchdog_exception`. This example will fail with a 401 error and display it on screen.

```

public function example2() {

  $msg = '';
  $client = \Drupal::httpClient();
  $uri = 'https://api.github.com/user';

  try {
    $request = $client->get($uri, [
      'auth' => ['username', 'password']
    ]);
    $response = $request->getBody();
    $msg .= "<br><strong>GET</strong>";
    $msg .= "<br>URI: " . $uri;
  }

  catch (ClientException $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  catch (\Exception $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $this->t($msg),
  ];

  return $build;
}

```

Guzzle Exceptions

You can get a full list of Exception types simply by listing the contents of the directory: \<drupal_root>/vendor/guzzlehttp/guzzle/src/Exception. Utilizing this list allows you to provide different behavior based on exception type.

At the time of writing, the contents of that directory is:

- BadResponseException.php
- ClientException.php - use this to handle a 4xx error
- ConnectException.php
- GuzzleException.php
- RequestException.php
- SeekException.php
- ServerException.php
- TooManyRedirectsException.php
- TransferException.php

HTTP response status codes

From <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

HTTP response status codes indicate whether a specific [HTTP](#) request has been successfully completed. Responses are grouped in five classes:

- 1 [Informational responses](#) (100 – 199)
- 2 [Successful responses](#) (200 – 299)

3 [Redirection messages](#) (300 – 399)

4 [Client error responses](#) (400 – 499)

5 [Server error responses](#) (500 – 599)

Reading from an API

In this example, a class was created which extends `SqlBase` (`docroot/core/modules/migrate/src/Plugin/migrate/source/SqlBase.php`). The code below is from the `prepareRow()` function which retrieves a row of data from the data source. In this case, rather than a SQL database, it is retrieved from an API. It uses basic http authentication during the GET call and if there are any errors, it updates a status elsewhere with a call to `setUpdStatus()`. The following code is not included below, but it may be interesting to know what it does. If the `GET` succeeds, the data is parsed out and put into variables to be returned to the called. This acts just like `prepareRow()` does when retrieving a row from a SQL source. Taxonomy terms are looked up and added if they don't already exist (so taxonomy term id's can be returned) and the status is updated showing this row was successfully retrieved.

```

$nard_auth_settings = Settings::get('nard_api_auth', []);
$uri = $nard_auth_settings['default']['server'] . ':' . $nard_auth_settings['default']['port'];
$uri .= '/sourcecontent/search';
$client = \Drupal::httpClient();

switch (strtolower($row->getSourceProperty('type'))) {
  case 'article':
    $uuid = $row->getSourceProperty('uuid');
    $id = $row->getSourceProperty('id');

    // Retrieve the article from the API.
    try {
      $response = $client->request('GET', $uri, [
        'auth' => [
          $nard_auth_settings['default']['username'],
          $nard_auth_settings['default']['password']
        ],
        'query' => [
          'uuid' => $row->getSourceProperty('uuid'),
          'properties' => 'Byline,updated,created,uuid',
        ],
        'timeout' => 3,
      ]);
    } catch (RequestException $e) {
      watchdog_exception('nard_myconnect', $e);
      \Drupal::logger('nard_myconnect')
        ->info("API error retrieving item $id with uuid=$uuid.");
      // TODO: Deal with this error
      $this->setUpdateStatus($uuid, 'error');
      return FALSE;
    }

    // Retrieve the details from the API call.
    try {
      $response = $response->getBody()->getContents();
    } catch (ClientException $e) {
      watchdog_exception('nard_myconnect', $e);
      \Drupal::logger('nard_myconnect')
        ->info("API error retrieving body for item $id with uuid=$uuid.");
      $this->setUpdateStatus($uuid, 'error');
      return FALSE;
    }

    // Parse retrieved JSON into fields.
    $response = json_decode($response, TRUE);
    if (!is_array($response)) {
      $response = [];
    }
  }
}

```

Curl Request using Drupal httpClient

From the now defunct link: <http://btobac.com/blog/how-do-curl-request-using-drupal-httpclient-drupal-8>

Here the author has an example of a function which takes a few parameters and can execute a POST, PUT or GET. There is no security code which you almost always need, but there is exception handling and error logging to Drupal watchdog.

Drupal HTTP client for curl HTTP request like POST, PUT, GET Method even for DELETE, you can add one type in the below switch case in the class method

```

class DrupalHTTPClient {

    public function initRequest($url, $headers = [], $content = "", $method = "POST", $msg = true, $type = "Third Party", $requestContentType = "json") {
        try {

            $client = \Drupal::httpClient();
            $params = ['http_errors' => FALSE];
            $params[$requestContentType] = $content;

            $params = array_merge($headers, $params);

            switch ($method) {
                case 'POST':
                    $response = $client->post($url, $params);
                    break;
                case 'PUT':
                    $response = $client->put($url, $params);
                    break;
                case 'GET':
                    $response = $client->get($url);
                    break;
                default:
                    $response = $client->post($url, $params);
                    break;
            }

            if ($response->getStatusCode() == '200') {
                $data = $response->getBody()->getContents();
                //Drupal::messenger()->addMessage(serialize($data));
                $result = json_decode($data, true);
                if($msg) {
                    $setMessage = is_array($result) && isset($result['message']) ? $result['message'] :
                    (is_string($data) ? $data : "Request is done successfully ");
                    \Drupal::messenger()->addMessage($type." ".$setMessage);
                }
                if(isset($result['exception'])){
                    $result_message = '';
                    foreach($result as $key => $value){
                        $result_message .= $key .': '.$value.', ';
                    }
                    \Drupal::logger('invalid_exception')->error(" \'Invalid Response: <i><strong>".$method."</strong> ".$url."</i> resulted in a <strong>'status:".$response->getStatusCode()." ". $response");
                    return NULL;
                }
                return $result;
            } else {
                \Drupal::messenger()->addError($type.' exception contact administrator');
                \Drupal::logger('server_exception')->error(" \'Server error: <i><strong>".$method."</strong> ".$url."</i> resulted in a <strong>'".$response->getStatusCode()." ".$response");
            }
        } catch (\GuzzleHttp\Exception\ConnectException $e) {
            \Drupal::messenger()->addError('Cannot contact '.$type);
            \Drupal::logger('connect_exception')->error(" \'Connection error: <i><strong>".$method."</strong> ".$url."</i> resulted in a <strong>'".$e->getMessage()."\'</strong> response");
        }
        return NULL;
    }

    $cl = new DrupalHTTPClient();
}

```

```
$url = "http://api.adadasdadadasd/adasd/ad/ad";
$content = ["key" => "value"];
$data = $cl->initRequest($url, [], $content, "POST", TRUE, "anytime just flag");
```

Download a file using guzzle

Details are borrowed from <https://gist.github.com/edutru1/9d04d7742545dbedd1a36f7b17632b7a>

```
public function example3() {

$msg = "";
$client = \Drupal::httpClient();
$uri = 'https://api.github.com/user';

try {

$source_uri = 'https://www.austinprogressivecalendar.com/sites/default/files/styles/medium/public/inserted-images/2018-04-02_5.jpg';
// Note sites/default/files/abc directory must exist for this to succeed.

$destination_uri = 'sites/default/files/abc/test.png';
/** @var \GuzzleHttp\Psr7\Response $response */
$response = $client->get($source_uri, ['sink' => $destination_uri]);
// file gets downloaded to /sites/default/files/abc/test.png

$msg .= "<br><strong>Retrieve File via Guzzle</strong>";
$msg .= "<br>Source: " . $source_uri;
$msg .= "<br>Dest: " . $destination_uri;
}

catch (ClientException $e) {
\Drupal::messenger()->addError($e->getMessage());
watchdog_exception('guzzle_examples', $e);
}

catch (\Exception $e) {
\Drupal::messenger()->addError($e->getMessage());
watchdog_exception('guzzle_examples', $e);
}

$build['content'] = [
'#type' => 'item',
'#markup' => $this->t($msg),
];

return $build;
}
```

Download a file using curl in PHP

For comparison, I've included an example of how to download a file using curl.

From [Stackoverflow: How to download a file using curl in php?](https://stackoverflow.com/questions/10147693/how-to-download-a-file-using-curl-in-php)

```
<?php

// Username or E-mail
$login = 'username';
// Password
$password = 'password';
// API Request
```

```

$url = 'https://example.com/api';
// POST data
$data = array('someTask', 24);
// Convert POST data to json
$data_string = json_encode($data);
// initialize cURL
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_USERPWD, "$login:$password");
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($ch, CURLOPT_POSTFIELDS, $data_string);
curl_setopt($ch, CURLOPT_HEADER, 1);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

// Execute cURL and store the response in a variable
$file = curl_exec($ch);

// Get the Header Size
$header_size = curl_getinfo($ch, CURLINFO_HEADER_SIZE);
// Get the Header from response
$header = substr($file, 0, $header_size);
// Get the Body from response
$body = substr($file, $header_size);
// Explode Header rows into an array
$header_items = explode("\n", $header);
// Close cURL handler
curl_close($ch);

// define new variable for the File name
$file_name = null;

// find the filename in the headers.
if(!preg_match('/filename="(.*)"/', $header, $matches)){
    // If filename not found do something...
    echo "Unable to find filename.<br>Please check the Response Headers or Header parsing!";
    exit();
} else {
    // If filename was found assign the name to the variable above
    $file_name = $matches[1];
}

// Check header response, if HTTP response is not 200, then display the error.
if(!preg_match('/200/', $header_items[0])){
    echo '<pre>'.print_r($header_items[0], true).'</pre>';
    exit();
} else {
    // Check header response, if HTTP response is 200, then proceed further.

    // Set the header for PHP to tell it, we would like to download a file
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Transfer-Encoding: binary');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Disposition: attachment; filename="'.$file_name');

    // Echo out the file, which then should trigger the download
    echo $file;
    exit();
}

```

```
    }  
}
```

Also there were other examples on that page that are probably worth looking at.

Someone responded with this shorter paste in November 2020 at <http://paste.debian.net/1170460/> with the following comment: this code is not downloading a file with PHP, it is PROXYING a file with PHP, and it's not doing a good job either, being way slower and more memory-hungry than required, this script would do it much faster, benchmark it

```
<?php  
  
declare(strict_types=1);  
$ch = curl_init();  
curl_setopt_array($ch, [  
    CURLOPT_URL => 'http://example.org',  
    CURLOPT_HEADERFUNCTION => function ($ch, string $header): int {  
        $header_trimmed = trim($header);  
        if (strlen($header_trimmed) > 0) {  
            header($header, FALSE);  
        }  
        return strlen($header);  
    }  
]);  
header('Content-Description: File Transfer');  
header('Content-Type: application/octet-stream');  
  
curl_exec($ch);  
curl_close($ch);
```

Resources

- Guzzle docs <https://docs.guzzlephp.org/en/stable/overview.html>
- Guzzle project <https://github.com/guzzle/guzzle>
- Article at Drupalize.me by William Hetherington from December 2015 on using Drupal 8 to speak http (using Guzzle): <https://drupalize.me/blog/201512/speak-http-drupal-httpclient>
- Nice little code snippet from J M Olivas showing how to use dependency injection with Guzzle at <https://gist.github.com/jmolivas/ca258d7f2742d9e1aae4>
- PSR-7: HTTP message interfaces describes common interfaces for representing HTTP messages and URIs <https://www.php-fig.org/psr/psr-7/>

[Back to top](#)

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

Page last modified: Apr 14 2023.

[Edit this page on GitHub](#)

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

Hooks

TABLE OF CONTENTS

- [Overview](#)
- [Modify the login form](#)
- [Modify the node edit form](#)
- [Modify fields in a node](#)
- [hook_update](#)
- [Theme hooks](#)
 - [Hook preprocess](#)
 - [hook preprocess node example 1](#)
 - [hook preprocess node example 2](#)
- [Organizing your hooks code the OOP way](#)
- [Reference](#)
 - [Entity hooks](#)
 - [Create operations](#)
 - [Read/Load operations](#)
 - [Save operations](#)
 - [Editing operations](#)
 - [Delete operations](#)
 - [View/render operations](#)
 - [Other entity hooks](#)
 - [.](#)
 - [Theme hooks](#)
 - [Overriding Theme Hooks](#)
 - [Preprocessing for Template Files](#)
 - [Theme hook suggestions](#)
 - [Altering theme hook suggestions](#)
 - [Reference Links](#)

views 172

Overview

Drupal hooks allow modules to alter and extend the behavior of Drupal core, or another module. They provide a way that code components in Drupal can communicate with one another. Using hooks, a module developer can change how core, or other modules work without changing the existing code. As a Drupal developer, understanding how to implement and invoke hooks is essential. (More at <https://drupalize.me/tutorial/what-are-hooks?p=2766>)

According to ChapGPT: Hooks are a key aspect of Drupal's module system, and allow developers to interact with the core functionality of Drupal 9 or Drupal 10. They can be used to alter or extend the behavior of Drupal's core features, such as adding custom validation to a form, changing the way content is displayed, or adding new actions to the administrative interface. Hooks provide a powerful and flexible way to customize Drupal to meet the needs of a specific project or site, without having to modify the core code. They are essential for developers who want to build custom modules or themes, and are a fundamental part of the Drupal development process.

Modify the login form

Here is code from hook_examples.module that modifies the user login form by adding a button. It passes the username and password that were entered to the mythical third party login endpoint.

```

/**
 * Implements hook_form_FORM_ID_alter().
 */
function hook_examples_form_user_login_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state) {
  // Add a custom submit button to the login form.
  $form['third_party_login'] = [
    '#type' => 'submit',
    '#value' => t('Login with Third-Party Provider'),
    '#submit' => ['hook_examples_user_login_form_submit'], // Call the custom submit function.
  ];
}

/**
 * Custom submit function for the login form.
 */
function hook_examples_user_login_form_submit(array &$form, \Drupal\Core\Form\FormStateInterface $form_state) {
  // Get the username and password from the form state.
  $username = $form_state->getValue('name');
  $password = $form_state->getValue('pass');

  // Build the URL for the third-party login page, including the username and password as query parameters.
  $login_url = 'https://thirdpartyprovider.com/login?username=' . urlencode($username) . '&password=' . urlencode($password);

  // Redirect the user to the third-party login page.
  $form_state->setRedirect($login_url);
}

```

In the above code, the `hook_examples_form_user_login_form_alter()` function implements the `hook_form_FORM_ID_alter()` hook, where `FORM_ID` is the ID of the form being altered, in this case `user_login_form`. The function modifies the login form by adding a custom submit button, with a submit handler function of `hook_examples_user_login_form_submit()`.

When the button is clicked, the `hook_examples_user_login_form_submit()` function gets the username and password from the form state , builds the URL for the third-party login page, including the username and password as query parameters. Finally, the user is redirected to this URL using the `$form_state->setRedirect()` method.

Modify the node edit form

In this example, the save button is changed from saying "save" to "update event"

```

use Drupal\Core\Form\FormStateInterface;
use Drupal\node\Entity\Node;

/**
 * Implements hook_form_alter().
 */
function hook_examples_form_alter(array &$form, FormStateInterface $form_state, $form_id) {
  if ($form_id === 'node_event_edit_form') {
    $node = $form_state->getFormObject()->getEntity();
    if ($node instanceof Node && $node->bundle() === 'event') {
      $form['actions']['submit']['#value'] = t('Update Event');
    }
  }
}

```

This code uses the `hook_form_alter` hook to alter the node edit form and modify the value of the submit button for nodes of type `event`. The `$form_id` argument is used to check if the form being altered is the node edit form for nodes of type `event`, and if it is, the submit button's value is changed to "Update Event". A redundant check is added to ensure the node is of type "event" for clarity.

Modify fields in a node

This example does all sorts of interesting things to the node as it is about to be saved.

It grabs some dates, fills out some fields if the user is anonymous, does some date calculations, changes the title of the node, looks up if to see if there is a connected node and grabs some info from it and updates a date field. Finally it invalidates some cache tags. Phew!

```
/*
 * Implements hook_ENTITY_TYPE_presave().
 */
function ogg_mods_node_presave(NodeInterface $node) {
  $type = $node->getType();
  if ($type == 'catalog_notice') {
    $end_date = NULL != $node->get('field_cn_start_end_dates')->end_value ? $node->get('field_cn_start_end_dates')->end_value : 'n/a';
    $govt_body = NULL != $node->field_cn_governmental_body->value ? $node->field_cn_governmental_body->value : 'Unnamed Government Body';
    $start_date_val = $node->get('field_cn_start_date')->value;

    $accountProxy = \Drupal::currentUser();
    $account = $accountProxy->getAccount();
    // Anonymous users automatically fill out the end_date.
    if (!$account->hasPermission('administer catalog notice')) {
      $days = (int) $node->get('field_cn_suspension_length')->value - 1;

      $end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
      $end_date->modify("+$days days");
      $end_date = $end_date->format("Y-m-d");
      $node->set('field_cn_end_date', $end_date);
    }

    // Always reset the title.
    $title = substr($govt_body, 0, 200) . " - $start_date_val";
    $node->setTitle($title);

    /*
     * Fill in Initial start and end dates if this is an extension of
     * a previously submitted notice.
     */
    $extension = $node->get('field_cn_extension')->value;
    if ($extension) {
      $previous_notice_nid = $node->get('field_cn_original_notice')->target_id;
      $previous_notice = Node::load($previous_notice_nid);
      if ($previous_notice) {
        $initial_start = $previous_notice->get('field_cn_start_date')->value;
        $initial_end = $previous_notice->get('field_cn_end_date')->value;
        $node->set('field_cn_initial_start_date', $initial_start);
        $node->set('field_cn_initial_end_date', $initial_end);
      }
    }
    $tags[] = 'ogg:node:' . $node->getType();
    $tags[] = 'ogg:node:' . $node->id();
    ogg_mods_invalidate_node($node);
    Cache::invalidateTags($tags);
  }

  /**
   * Invalidate cache associated with various nodes.
   *
   * @param \Drupal\node\NodeInterface $node
   *
   * @throws \Drupal\Component\Plugin\Exception\InvalidPluginDefinitionException
   * @throws \Drupal\Component\Plugin\Exception\PluginNotFoundException
   */
}
```

```

*/
function ogg_mods_invalidate_node(NodeInterface $node) {
  $tags = [];
  $node_type = $node->getType();
  $tags[] = 'ogg:node:' . $node->id();
  switch ($node_type) {
    case 'news':
      $tags[] = 'ogg:node:home';
      $tags[] = 'ogg:node:landing_page_news';
      $tags[] = 'ogg:node:news';
      $tags[] = 'ogg:views:home_recent_news';
      break;
    default:
      break;
  }
  if (!empty($tags)) {
    Cache::invalidateTags($tags);
  }
}

```

You can read more about using hook_entity_presave at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/function/hook_entity_presave/10 as well as

hook_ENTITY_TYPE_presave at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/function/hook_ENTITY_TYPE_presave/10

hook_update

Almost every project that runs for a while will require some hook_updates. This is the facility used to do automated changes to your sites. Here is an example that updates a menu item with a title 'Support'.

This code comes from a .install file:

```

function partridge_update_8002() {

  $mids = \Drupal::entityQuery('menu_link_content')
    ->condition('menu_name', 'part-wide-utility')
    ->execute();

  foreach($mids as $mid) {
    $menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($mid);

    $title = $menu_link->getTitle();
    if ($title === 'Support') {
      $menu_link->set('weight', 2);
      $menu_link->set('expanded', TRUE);
      $menu_link->set('link', 'https://www.google.com');
      $menu_link->save();
    }
  }
}

```

Theme hooks

Here is an excerpt from the Theme System Overview at

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/themeable/10>:

Preprocessing for Template Files

Several functions are called before the template file is invoked to modify the variables that are passed to the template. These make up the

"preprocessing" phase, and are executed (if they exist), in the following order (note that in the following list, HOOK indicates the hook being called or a less specific hook. For example, if '#theme'=>'node__article' is called, hook is node__article and node.

MODULE indicates a module name, THEME indicates a theme name, and ENGINE indicates a theme engine name). Modules, themes, and theme engines can provide these functions to modify how the data is preprocessed, before it is passed to the theme template:

- **template preprocess(&\$variables, \$hook)**: Creates a default set of variables for all theme hooks with template implementations. Provided by Drupal Core.
- **template preprocess_HOOK(&\$variables)**: Should be implemented by the module that registers the theme hook, to set up default variables.
- **MODULE preprocess(&\$variables, \$hook)**: hook_preprocess() is invoked on all implementing modules.
- **MODULE preprocess_HOOK(&\$variables)**: hook_preprocess_HOOK() is invoked on all implementing modules, so that modules that didn't define the theme hook can alter the variables.
- **ENGINE engine preprocess(&\$variables, \$hook)**: Allows the theme engine to set necessary variables for all theme hooks with template implementations.
- **ENGINE engine preprocess_HOOK(&\$variables)**: Allows the theme engine to set necessary variables for the particular theme hook.
- **THEME preprocess(&\$variables, \$hook)**: Allows the theme to set necessary variables for all theme hooks with template implementations.
- **THEME preprocess_HOOK(&\$variables)**: Allows the theme to set necessary variables specific to the particular theme hook.

Hook preprocess

Generally, .theme files will include the following to create or alter variables for :

- **hook preprocess_html()** the html template
- **hook preprocess_page()** the page template
- **hook preprocess_node()** the node template. Note hook_node_type preprocess_node() also works where you can specify the node type e.g. wc_product preprocess_node() which expects a content type of wc_product.

hook preprocess_node example 1

To add a custom variable to be displayed in your template, add a function in your .theme file like the one listed below. This example also adds a #suffix to the field_image which renders that string after the field_image is rendered.

```
function mytheme preprocess_node(&$variables) {  
  $variables['custom_variable'] = "Bananas are yellow";  
  $variables['content']['field_image'][#suffix] = "this suffix on the image";  
  kint($variables);  
}
```

The render array you want to change will be in the content variable which shows up in the kint output as \$variables['content']

Usually fields such as field_image will be automatically rendered by the node template (unless you've tweaked it to display in some other template.)

In your node's Twig template you would specify `` to have it display on every node.

hook preprocess_node example 2

This code is used to make a date range like 3/30/2023 -- 3/31/2023 appear as Mar 30-31, 2023

The date values are stored in the field_date which is a date range field. This code is from the .theme file. Here we retrieve the starting and ending date values:

```
$from = $variables['node']->get('field_date')->getValue()[0]['value'];  
$to = $variables['node']->get('field_date')->getValue()[0]['end_value'];
```

Here is the hook preprocess_node()

Notice that we are creating a Twig variable called "scrunch_date" which we want to display.

```
use Drupal\Core\Datetime\DrupalDateTime;

/**
 * Implements hook_preprocess_node
 *
 * @param $variables
 */
function veryst_preprocess_node(&$variables) {
  if (!empty($variables['content']['field_date'])) {
    $date = $variables['content']['field_date'];

    $from = new DrupalDateTime($variables["node"]->get('field_date')->getValue()[0]['value']);
    $date_array = explode("-", $from);
    $from_day = substr($date_array[2], 0, 2);
    $from_month = $date_array[1];

    $to = new DrupalDateTime($variables["node"]->get('field_date')->getValue()[0]['end_value']);
    $date_array = explode("-", $to);
    $to_day = substr($date_array[2], 0, 2);
    $to_month = $date_array[1];

    if ($from_month === $to_month && $from_day != $to_day) {
      $variables['scrunch_date'] = [
        '#type' => 'markup',
        '#markup' => $from->format("M j") . $to->format("], Y"),
      ];
    }
  }
  // kint($variables);
}
```

Now in the twig template we can output the scrunch_date we created in the template file: `web/themes/mytheme/templates/node/node--seminar-teaser.html.twig`

Organizing your hooks code the OOP way

from [Drupal 8 How to organise your hooks code in classes \(Object-oriented way\)](#).

To implement a hook example like `hook_form_FORM_ID_alter` for `node_article_edit_form`.

So instead of do something like:

```

use Drupal\Core\Form\FormStateInterface;

/**
 * Implements hook_form_BASE_FORM_ID_alter().
 */
function MY_MODULE_form_node_article_edit_form_alter(&$form, FormStateInterface $form_state, $form_id) {
  // Your code here the two following lines just an examples.
  // Hide some fields.
  $form['field_SOME_FIELD_NAME']['#access'] = FALSE;
  // Attach some library ....
  $form['#attached']['library'][] = 'MY_MODULE/SOME_LIBRARY';
}

```

You can create a class called **NodeArticleEditFormHandler** inside your **src** folder like the following:

```

<?php

namespace Drupal\MY_MODULE;

use Drupal\Core\Form\FormStateInterface;

/**
 * Class NodeArticleEditFormHandler
 *
 * @package Drupal\MY_MODULE
 */
class NodeArticleEditFormHandler {

  /**
   * Alter Form.
   *
   * @param array $form
   *   Form array.
   * @param \Drupal\Core\Form\FormStateInterface $form_state
   *   The current state of the form.
   * @param string $form_id
   *   String representing the id of the form.
   */
  public function alterForm(array &$form, FormStateInterface $form_state, $form_id) {
    // Your code here the two following lines just an examples.
    // Hide some fields.
    $form['field_SOME_FIELD_NAME']['#access'] = FALSE;
    // Attach some library ....
    $form['#attached']['library'][] = 'MY_MODULE/SOME_LIBRARY';
  }
}

```

In case you need other services you can inject your dependencies by make your class [implements ContainerInjectionInterface](#) here is an example with current user service injection:

```

<?php

namespace Drupal\MY_MODULE;

use Drupal\DependencyInjection\ContainerInjectionInterface;
use Drupal\Core\Form\FormStateInterface;

```

```

use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Class NodeArticleEditFormHandler
 *
 * @package Drupal\MY_MODULE
 */
class NodeArticleEditFormHandler implements ContainerInjectionInterface {

 /**
 * The current user account.
 *
 * @var \Drupal\Core\Session\AccountProxyInterface
 */
protected $currentUser;

 /**
 * NodeArticleEditFormHandler constructor.
 *
 * @param \Drupal\Core\Session\AccountProxyInterface $current_user
 *   The current user.
 */
public function __construct(AccountProxyInterface $current_user) {
    $this->currentUser = $current_user;
}

 /**
 * @inheritDoc
 */
public static function create(ContainerInterface $container) {
    return new static(
        $container->get('current_user')
    );
}

 /**
 * Alter Form.
 *
 * @param array $form
 *   Form array.
 * @param \Drupal\Core\Form\FormStateInterface $form_state
 *   The current state of the form.
 * @param string $form_id
 *   String representing the id of the form.
 */
public function alterForm(array &$form, FormStateInterface $form_state, $form_id) {
    // Example to get current user.
    $currentUser = $this->currentUser;
    // Your code here the two following lines just an examples.
    // Hide some fields.
    $form['field_SOME_FIELD_NAME']['#access'] = FALSE;
    // Attach some library ....
    $form['#attached']['library'][] = 'MY_MODULE/SOME_LIBRARY';
}

}

```

And after that change your hook into:

```

use Drupal\MY_MODULE\NodeArticleEditFormHandler;

/**
 * Implements hook_form_BASE_FORM_ID_alter().
 */
function MY_MODULE_form_node_article_edit_form_alter(&$form, FormStateInterface $form_state, $form_id) {
  return \Drupal::service('class_resolver')
    ->getInstanceFromDefinition(NodeArticleEditFormHandler::class)
    ->alterForm($form, $form_state, $form_id);
}

```

We are done! Now your .module file is more clean, readable and maintainable with less code. You can do that with every hook for instance **EntityHandler** like:

```

use Drupal\MY_MODULE\EntityHandler;

/**
 * Implements hook_entity_presave().
 */
function MY_MODULE_entity_presave(EntityInterface $entity) {
  return \Drupal::service('class_resolver')
    ->getInstanceFromDefinition(EntityHandler::class)
    ->entityPresave($entity);
}

```

And so on! You can [see another example in Drupal core from the content moderation form](#)

Reference

Entity hooks

Here is an excerpt from the Drupal API at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/group/entity_crud/10:

CREATE OPERATIONS

To create an entity:

```

$entity = $storage->create();
// Add code here to set properties on the entity. // Until you call save(), the entity is just in memory.
$entity->save();

```

There is also a shortcut method on entity classes, which creates an entity with an array of provided property values: \Drupal\Core\Entity::create().

Hooks invoked during the create operation:

- [hook_ENTITY_TYPE_create\(\)](#)
- [hook_entity_create\(\)](#)
- When handling content entities, if a new translation is added to the entity object:
 - [hook_ENTITY_TYPE_translation_create\(\)](#)
 - [hook_entity_translation_create\(\)](#)

See [Save operations](#) below for the save portion of the operation.

READ/LOAD OPERATIONS

To load (read) a single entity:

```

$entity = $storage->load($id);
To load multiple entities: $entities = $storage->loadMultiple($ids);

```

Since load() calls loadMultiple(), these are really the same operation. Here is the order of hooks and other operations that take place during entity loading:

- Entity is loaded from storage.
- postLoad() is called on the entity class, passing in all of the loaded entities.
- [hook_entity_load\(\)](#)
- [hook_ENTITY_TYPE_load\(\)](#)

When an entity is loaded, normally the default entity revision is loaded. It is also possible to load a different revision, for entities that support revisions, with this code:

```
$entity = $storage->loadRevision($revision_id);
```

This involves the same hooks and operations as regular entity loading.

The "latest revision" of an entity is the most recently created one, regardless of it being default or pending. If the entity is translatable, revision translations are not taken into account either. In other words, any time a new revision is created, that becomes the latest revision for the entity overall, regardless of the affected translations. To load the latest revision of an entity:

```
$revision_id = $storage->getLatestRevisionId($entity_id);  
  
$entity = $storage->loadRevision($revision_id);
```

As usual, if the entity is translatable, this code instantiates into \$entity the default translation of the revision, even if the latest revision contains only changes to a different translation:

```
$is_default = $entity->isDefaultTranslation(); // returns TRUE
```

The "latest translation-affected revision" is the most recently created one that affects the specified translation. For example, when a new revision introducing some changes to an English translation is saved, that becomes the new "latest revision". However, if an existing Italian translation was not affected by those changes, then the "latest translation-affected revision" for Italian remains what it was. To load the Italian translation at its latest translation-affected revision:

```
$revision_id = $storage->getLatestTranslationAffectedRevisionId($entity_id, 'it'); $it_translation = $storage->loadRevision($revision_id)->getTranslation('it');
```

SAVE OPERATIONS

To update an existing entity, you will need to load it, change properties, and then save; as described above, when creating a new entity, you will also need to save it. Here is the order of hooks and other events that happen during an entity save:

- preSave() is called on the entity object, and field objects.
- [hook_ENTITY_TYPE_presave\(\)](#)
- [hook_entity_presave\(\)](#)
- Entity is saved to storage.
- For updates on content entities, if there is a translation added that was not previously present:
 - [hook_ENTITY_TYPE_translation_insert\(\)](#)
 - [hook_entity_translation_insert\(\)](#)
- For updates on content entities, if there was a translation removed:
 - [hook_ENTITY_TYPE_translation_delete\(\)](#)
 - [hook_entity_translation_delete\(\)](#)
- postSave() is called on the entity object.
- [hook_ENTITY_TYPE_insert\(\)](#) (new) or [hook_ENTITY_TYPE_update\(\)](#) (update)
- [hook_entity_insert\(\)](#) (new) or [hook_entity_update\(\)](#) (update)

Some specific entity types invoke hooks during preSave() or postSave() operations. Examples:

- **Field configuration preSave()**: hook_field_storage_config_update_forbid()

- **Node postSave()**: hook_node_access_records() and hook_node_access_records_alter()
- Config entities that are acting as entity bundles in postSave(): hook_entity_bundle_create()
- **Comment**: hook_comment_publish() and hook_comment_unpublish() as appropriate.

Note that all translations available for the entity are stored during a save operation. When saving a new revision, a copy of every translation is stored, regardless of it being affected by the revision.

EDITING OPERATIONS

When an entity's add/edit form is used to add or edit an entity, there are several hooks that are invoked:

- [hook_entity_prepare_form\(\)](#)
- [hook_ENTITY_TYPE_prepare_form\(\)](#)
- [hook_entity_form_display_alter\(\)](#) (for content entities only)

DELETE OPERATIONS

To delete one or more entities, load them and then delete them:

```
$entities = $storage->loadMultiple($ids);
$storage->delete($entities);
```

During the delete operation, the following hooks and other events happen:

- preDelete() is called on the entity class.
- [hook_ENTITY_TYPE_predelete\(\)](#)
- [hook_entity_predelete\(\)](#)
- Entity and field information is removed from storage.
- postDelete() is called on the entity class.
- [hook_ENTITY_TYPE_delete\(\)](#)
- [hook_entity_delete\(\)](#)

Some specific entity types invoke hooks during the delete process. Examples:

- **Entity bundle postDelete()**: hook_entity_bundle_delete()

Individual revisions of an entity can also be deleted:

```
$storage->deleteRevision($revision_id);
```

This operation invokes the following operations and hooks:

- Revision is loaded (see [Read/Load operations](#) above).
- Revision and field information is removed from the database.
- [hook_ENTITY_TYPE_revision_delete\(\)](#)
- [hook_entity_revision_delete\(\)](#)

VIEW/RENDER OPERATIONS

To make a render array for a loaded entity:

```
// You can omit the language ID if the default language is being used.
```

```
$build = $view_builder
->view($entity, 'view_mode_name', $language->getId());
```

You can also use the viewMultiple() method to view multiple entities.

Hooks invoked during the operation of building a render array:

- [hook_entity_view_mode_alter\(\)](#)
- [hook_ENTITY_TYPE_build_defaults_alter\(\)](#)
- [hook_entity_build_defaults_alter\(\)](#)

View builders for some types override these hooks, notably:

- The Tour view builder does not invoke any hooks.
- The Block view builder invokes hook_block_view_alter() and hook_block_view_BASE_BLOCK_ID_alter(). Note that in other view builders, the view alter hooks are run later in the process.

During the rendering operation, the default entity viewer runs the following hooks and operations in the pre-render step:

- [hook_entity_view_display_alter\(\)](#)
- [hook_entity_prepare_view\(\)](#)
- Entity fields are loaded, and render arrays are built for them using their formatters.
- [hook_entity_display_build_alter\(\)](#)
- [hook_ENTITY_TYPE_view\(\)](#)
- [hook_entity_view\(\)](#)
- [hook_ENTITY_TYPE_view_alter\(\)](#)
- [hook_entity_view_alter\(\)](#)

Some specific builders have specific hooks:

- The Node view builder invokes hook_node_links_alter().
- The Comment view builder invokes hook_comment_links_alter().

After this point in rendering, the theme system takes over. See the [Theme system and render API topic](#) for more information.

OTHER ENTITY HOOKS

Some types of entities invoke hooks for specific operations:

- Searching nodes:
 - [hook_ranking\(\)](#)
 - Query is executed to find matching nodes
 - Resulting node is loaded
 - Node render array is built
 - [comment_node_update_index\(\)](#) is called (this adds "N comments" text)
 - [hook_node_search_result\(\)](#)
- Search indexing nodes:
 - Node is loaded
 - Node render array is built
 - [hook_node_update_index\(\)](#)

Theme hooks

Here is an excerpt from the Theme System Overview at

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/themeable/10>:

The theme system is invoked in [\Drupal\Core\Render\Renderer::doRender\(\)](#) by calling the [\Drupal\Core\Theme\ThemeManagerInterface::render\(\)](#) function, which operates on the concept of "theme hooks". Theme hooks define

how a particular type of data should be rendered. They are registered by modules by implementing `hook_theme()`, which specifies the name of the hook, the input "variables" used to provide data and options, and other information. Modules implementing `hook_theme()` also need to provide a default implementation for each of their theme hooks in a Twig file, and they may also provide preprocessing functions. For example, the core Search module defines a theme hook for a search result item in `search_theme()`:

```
return array(
  'search_result' => array(
    'variables' => array(
      'result' => NULL,
      'plugin_id' => NULL,
    ),
    'file' => 'search.pages.inc',
  ),
);
```

Given this definition, the template file with the default implementation is [search-result.html.twig](#), which can be found in the `core/modules/search/templates` directory, and the variables for rendering are the search result and the plugin ID. In addition, there is a function `template_preprocess_search_result()`, located in file [search.pages.inc](#), which preprocesses the information from the input variables so that it can be rendered by the Twig template; the processed variables that the Twig template receives are documented in the header of the default Twig template file.

OVERRIDING THEME HOOKS

Themes may register new theme hooks within a `hook_theme()` implementation, but it is more common for themes to override default implementations provided by modules than to register entirely new theme hooks. Themes can override a default implementation by creating a template file with the same name as the default implementation; for example, to override the display of search results, a theme would add a file called [search-result.html.twig](#) to its templates directory. A good starting point for doing this is normally to copy the default implementation template, and then modifying it as desired.

PREPROCESSING FOR TEMPLATE FILES

Several functions are called before the template file is invoked to modify the variables that are passed to the template. These make up the "preprocessing" phase, and are executed (if they exist), in the following order (note that in the following list, HOOK indicates the hook being called or a less specific hook. For example, if '#theme'=>'node__article' is called, hook is node__article and node. MODULE indicates a module name, THEME indicates a theme name, and ENGINE indicates a theme engine name). Modules, themes, and theme engines can provide these functions to modify how the data is preprocessed, before it is passed to the theme template:

- **[template preprocess\(&\\$variables, \\$hook\)](#)**: Creates a default set of variables for all theme hooks with template implementations. Provided by Drupal Core.
- **[template_preprocess_HOOK\(&\\$variables\)](#)**: Should be implemented by the module that registers the theme hook, to set up default variables.
- **[MODULE_preprocess\(&\\$variables, \\$hook\)](#)**: `hook_preprocess()` is invoked on all implementing modules.
- **[MODULE_preprocess_HOOK\(&\\$variables\)](#)**: `hook_preprocess_HOOK()` is invoked on all implementing modules, so that modules that didn't define the theme hook can alter the variables.
- **[ENGINE_engine_preprocess\(&\\$variables, \\$hook\)](#)**: Allows the theme engine to set necessary variables for all theme hooks with template implementations.
- **[ENGINE_engine_preprocess_HOOK\(&\\$variables\)](#)**: Allows the theme engine to set necessary variables for the particular theme hook.
- **[THEME_preprocess\(&\\$variables, \\$hook\)](#)**: Allows the theme to set necessary variables for all theme hooks with template implementations.
- **[THEME_preprocess_HOOK\(&\\$variables\)](#)**: Allows the theme to set necessary variables specific to the particular theme hook.

THEME HOOK SUGGESTIONS

In some cases, instead of calling the base theme hook implementation (either the default provided by the module that defined the hook, or the override provided by the theme), the theme system will instead look for "suggestions" of other hook names to look for. Suggestions can be specified in several ways:

- In a render array, the '#theme' property (which gives the name of the hook to use) can be an array of theme hook names instead of a single hook name. In this case, the render system will look first for the highest-priority hook name, and if no implementation is found, look for the second, and so on. Note that the highest-priority suggestion is at the end of the array.

- In a render array, the '#theme' property can be set to the name of a hook with a '__SUGGESTION' suffix. For example, in search results theming, the hook 'item_list__search_results' is given. In this case, the render system will look for theme templates called [item-list-search-results.html.twig](#), which would only be used for rendering item lists containing search results, and if this template is not found, it will fall back to using the base [item-list.html.twig](#) template. This type of suggestion can also be combined with providing an array of theme hook names as described above.
- A module can implement hook_theme_suggestions_HOOK(). This allows the module that defines the theme template to dynamically return an array containing specific theme hook names (presumably with '__' suffixes as defined above) to use as suggestions. For example, the Search module does this in search_theme_suggestions_search_result() to suggest search_result_PLUGIN as the theme hook for search result items, where PLUGIN is the machine name of the particular search plugin type that was used for the search (such as node_search or user_search).

For further information on overriding theme hooks see <https://www.drupal.org/node/2186401>

ALTERING THEME HOOK SUGGESTIONS

Modules can also alter the theme suggestions provided using the mechanisms of the previous section. There are two hooks for this: the theme-hook-specific hook_theme_suggestions_HOOK_alter() and the generic hook_theme_suggestions_alter(). These hooks get the current list of suggestions as input, and can change this array (adding suggestions and removing them).

Reference Links

- [What are hooks? from Drupalize.me March 2022](#)
 - [Theme system overview on api.drupal.org](#)
 - [How to organize your hooks the object oriented way by Azz-eddine BERRAMOU Mar 2020](#)
-
-

[Back to top](#)

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

Page last modified: May 14 2023.

[Edit this page on GitHub](#)

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

Learning and keeping up with Drupal

TABLE OF CONTENTS

- [Free videos](#)
- [Blogs and articles](#)
- [Pay videos](#)
- [Drupal Training](#)
- [Keep up with Drupal news](#)
- [Drupal Podcasts](#)
- [Books](#)

views 167

Free videos

- Saranya Ashokkumar is a prolific youtube content creator. Check out some nice short form videos on how to use various modules, create modules etc. <https://www.youtube.com/@d4drupal324/featured>

Blogs and articles

- Phil Norton of #!code writes some amazing articles at <https://www.hashbangcode.com/> and has useful code snippets at <https://www.hashbangcode.com/snippets>
- Martin Anderson-Clutz's sandbox for a wide variety of fantastic Drupal, Acquia and module information at <https://www.mandclu.com/>. He is also the author and maintainer of the [Smart Date Module](#) and about 8 bazillion other modules.
- Matt Glaman posts regularly about fascinating aspects of Drupal (and is the author of existing and upcoming Drupal books). Keep up with him at <https://mglaman.dev/>

Pay videos

- [drupalize.me](#) is an amazing training facility spun off from [Lullabot](#) which provides on-demand training and articles on all things Drupal. This is definitely worth a subscription if you want to fully grasp Drupal development or bring new people up to speed.
- [SymfonyCasts](#) is another incredibly good service that teaches all about the nuts and bolts of Symfony, PHPUnit and many other tools. I think the Drupal course they have might be a little dated now.

Drupal Training

- Mike Anello of Drupal Easy has some training courses worth checking out at <https://www.drupaleeasy.com/academy>. These include: Drupal Career Online and Professional Module Development. Mike also offers project coaching and consulting.

Keep up with Drupal news

Everything used to happen on IRC, but now most of that seems to be on Slack at <https://drupal.slack.com/>

Other great sources of news are:

- Planet drupal at <https://www.drupal.org/planet>
- The Weekly Drop, a Drupal newsletter The Drupal Association has partnered with TheWeeklyDrop to bring up to date News and Events to the Drupal community. Free subscription at <http://www.theweeklydrop.com/>
- TheDropTimes is a news website started with the vision of contributing to the growth of a vibrant community of users and contributors around Drupal through the process of covering and promoting everything happening around Drupal. <https://www.thedropstimes.com/>
- Feed of Drupal related articles from all over the web <https://drupalsun.com>
- Mastodon is starting to show signs of Drupal life with a Mastodon instance focused on Drupal at <https://drupal.community/explore>
- Twitter has many useful Drupal posts - https://twitter.com/search?q=%23drupal&src=typed_query

Drupal Podcasts

- Talking Drupal is a long standing weekly conversation about and around Drupal development. It was started by Stephen Cross, John Picozzi, and Nic Lafin in 2013 and is still going strong. Check it out at <https://www.talkingdrupal.com>
- Lullabot Podcast: News, interviews, and tips about Drupal and Open Source from the team at Lullabot.
<https://www.lullabot.com/podcasts/lullabot-podcast>
- Mike Anello has a Drupal Easy podcast at <https://www.drupaleeasy.com/podcast>

Books

There are many books about Drupal that are worth checking out.

- Daniel Sipos's Drupal 10 Module Development: Develop and deliver engaging and intuitive enterprise-level apps, 4th Edition.
<https://www.packtpub.com/product/drupal-10-module-development-fourth-edition/9781837631803>
- Adam Bergstein's Drupal 10 Masterclass: Build responsive Drupal applications to deliver custom and extensible digital experiences to users. https://www.amazon.com/Drupal-Masterclass-responsive-applications-experiences-ebook/dp/B0BNNS7JCM/ref=d_pd_sbs_vft_none_sccl_1_1/146-2345249-6494660
- Matt Glaman and Kevin Quillen's Drupal 10 Development Cookbook. Published in Feb 2023. <https://www.packtpub.com/product/drupal-10-development-cookbook-third-edition/9781803234960>
- Daniel Sipos's Drupal 9 Module Development Third Edition Published in Aug 2020. <https://www.packtpub.com/product/drupal-9-module-development-third-edition/9781800204621>
- Fran Gil's Expert in Drupal 9 Front-End Development. Updated in Sep 2022 <https://www.forcontu.com/en/books/expert-in-drupal-9-front-end-development>
- Fran Gil's Expert in Drupal 9 Back-End Development. Updated in Sep 2022 <https://www.forcontu.com/en/books/expert-in-drupal-9-back-end-development>
- Check out more older Drupal books from Packt publishing at <https://subscription.packtpub.com/search?query=Drupal>
- Drupal Book.org is an online only book by Ivan. Updated Sep 2018. While it is not complete, it is a valuable resource.
<https://drupalbook.org/>

[Back to top](#)

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

Page last modified: May 8 2023.

[Edit this page on GitHub](#)

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

Links, Aliases and URLs

TABLE OF CONTENTS

- [Create an external url](#)
- [Create an internal url](#)
- [The Drupal Core Url Class](#)
- [The Drupal Core Link Class](#)
 - [Create a link to a node](#)
 - [Create a link to a path with parameters](#)
- [Another way to create a link to a node:](#)
- [Create a link from an internal URL](#)
- [Check if a link field is empty](#)
- [Retrieve a link field from a node or a paragraph](#)
- [Retrieve a URL field](#)
 - [External links](#)
 - [Internal links](#)
- [Get the NID from a URL Alias](#)
- [Get the Taxonomy Term ID from a URL alias](#)
- [Get URL alias for a taxonomy term](#)
- [Get the User ID from a URL alias](#)
- [Get the URL alias for a node](#)
- [Create a Node Alias](#)
- [Get the current Path](#)
- [Get current nid, node type and title](#)
- [How to get current Route name](#)
- [Get current Document root path](#)
- [Retrieve URL argument parameters](#)
- [Retrieve query and GET or POST parameters \(\\$_POST and \\$_GET\)](#)
- [Modify URL Aliases programmatically with hook_pathauto_alias_alter](#)
- [Drupal l\(\) is deprecated](#)
- [Reference links](#)

views 98

Create an external url

```
use Drupal\Core\Url  
$url = Url::fromUri('http://testsite.com/go/here');
```

Create an internal url

First a simple URL:

```

use Drupal\Core\Url;

$url = Url::fromUri('internal:/reports/search');

// Or.

$url = Url::fromUri('internal:/dashboard/achievements')

// Or Using link generator to create a GeneratedLink.

$url = Url::fromUri('internal:/node/1');

$link = \Drupal::service('link_generator')->generate('My link', $url);

// OR

$url = Url::fromUri('internal:/');

// OR

'url' => Url::fromRoute('<front>'),

// OR

'url' => Url::fromRoute('hello_world.hello'),

```

Then something more complicated like this URL to /reports/search?user=admin

```

$options = [
  'query' => ['user' => 'admin'],
];
$url = Url::fromUri('internal:/reports/search', $options);

```

The Drupal Core Url Class

The `Drupal\Core\Url` class is often used to create URL's. Two important methods are:

`url::fromRoute()` which takes a route name and parameters and

`url::fromUri()` which takes an internal or external URL

See how these are used in some of the examples below.

The Drupal Core Link Class

Closely related and often used in conjunction with the Drupal Core URL class is the `Drupal\Core\Link` class.

You can generate links several different ways.

Create a link to a node

```

//Using link generator to create a GeneratedLink.

$url = Url::fromUri('internal:/node/1');

$link = \Drupal::service('link_generator')->generate('My link', $url);

```

Create a link to a path with parameters

To create a link to a path like /reports/search?user=admin use this code.

```

$option = [
  'query' => ['user' => 'admin'],
];
$url = Url::fromUri('internal:/reports/search', $option);

// use the Link class.
$link = Link::fromTextAndUrl('My link', $url);
$renderable_array = $link->toRenderable();
return $renderable_array;

```

Another way to create a link to a node:

```

$nid = $item->id();
$options = ['relative' => TRUE]; //could be absolute instead of relative.

$url = Url::fromRoute('entity.node.canonical',[node' => $nid], $options);
$link = \Drupal::service('link_generator')->generate('My link', $url);

```

Create a link from an internal URL

```

use Drupal\Core\Url

$url = Url::fromUri('internal:/reports/search');
$link = \Drupal::service('link_generator')->generate('My link', $url);

// ->toString() will extract the string of the URL.
$url_string = Url::fromUri('internal:/node/' . $id)->toString();

```

Check if a link field is empty

```

if (!$citation_node->field_link->uri) {
  // Empty.
}

```

Retrieve a link field from a node or a paragraph

The link field `field_link` is extracted from the node and a valid uri is extracted from that field.

```

$correction_node = Node::load($nid);
$current_url = $correction_node->get('field_link')->uri;

```

Or from a paragraph field

```

use Drupal\paragraphs\Entity\Paragraph;

$para = Paragraph::load($target_id);
$link = $para->field_link;
$link_uri = $para->field_link->uri;

```

Or a more convoluted example that extracts the url string for display from a link field.

```

if ($sf_contract) {
  // first() returns a Drupal\link\Plugin\FieldType\LinkItem
  $vendor_url = $sf_contract->field_vendor_url->first();
  if ($vendor_url) {
    // returns a Drupal\Core\Url.
    $vendor_url = $vendor_url->getUrl();
    $vendor_url_string = $vendor_url->toString();
  }
}

```

Removing `->first()` as in:

```
$vendor_url = $sf_contract->field_vendor_url;
```

returns a `Drupal\Core\Field\FieldItemList` which is a list of fields so you then would have to pull out the first field and extract the URI out of that. I'm not sure why Drupal considers it multiple values instead of just one. This was not set up as a multivalue field.

Retrieve a URL field

External links

You can get the URL (for external links) and then just the text part.

Note this doesn't work for internal links. Note also this slightly convoluted example has a reference field `field_sf_contract_ref` which has a link to another entity and the `field_vendor_url->first()->getUrl()` is the important part. Also note, this is a single-value field (not a multivalue field – so the `first()` call may be a little disturbing to those who expect things to be a little clearer.)

```

$vendor_url = $node->field_sf_contract_ref->entity->field_vendor_url->first()->getUrl();
if ($vendor_url) {
  $vendor_url = $vendor_url->getUri();
  //OR
  $vendor_url = $vendor_url->toString();
}

```

A slightly simpler example from a form

```

$citation_link = $citation->get('field_link');
if (!$citation_link->isEmpty()) {
  $citation_link = $citation->field_link->first()->getUrl()->toString();
}

```

Internal links

For internal links, use `getUrl()`for the URL and `->title` for the title.

```

$instructions_node = Node::load($order_type_instructions_nid);
if ($instructions_node) {
  $order_link = $instructions_node->field_link->first();
  if ($order_link) {
    $uri = $order_link->uri;
    $variables['order_link_title'] = $order_link->title;
    $order_url = $order_link->getUrl();
    if ($order_url) {
      $variables['order_type_link'] = $order_url;
    }
  }
}

```

Get the NID from a URL Alias

To get the nid for a node, you can pass the URL alias to getPathByAlias.

```
// Given "/test-node, returns "/node/32".
$alias = "/test-node";
$path = \Drupal::service('path_alias.manager')->getPathByAlias($alias);
```

OR

If you have a URL for a node and you want its nid

```
$route = $url->getRouteParameters();
// first check if it's a node.
if (isset($route['node'])) {
  $nid = $route['node'];
}
```

Get the Taxonomy Term ID from a URL alias

Returns taxonomy/term/5

```
$term_path_with_tid = \Drupal::service('path_alias.manager')->getPathByAlias('/hunger-strike');
```

Get URL alias for a taxonomy term

This returns term/5 if no alias is set, otherwise it returns the alias.

```
$term5_url = Url::fromRoute('entity.taxonomy_term.canonical', ['taxonomy_term' => 5], $options);
$term5_alias = $term5_url->toString();
```

Get the User ID from a URL alias

Returns "/user/2"

```
//User
$user_path_with_uid = \Drupal::service('path_alias.manager')->getPathByAlias('/selwyn-the-chap');
```

Get the URL alias for a node

If no alias is set, this will return "/node/32". Note. If there are multiple aliases, you will get the most recently created one.

```
$node_path = '/node/32';
$node32_alias = \Drupal::service('path_alias.manager')->getAliasByPath($node_path);
```

Use this code if you need the absolute URL . If node/32 has a URL alias set to "/test-node" it returns "https://d9book2.ddev.site/test-node" . If you specify absolute => FALSE, it returns "/test-node" .

```
use Drupal\Core\Url;
// Note. If a pathauto url alias is not set, it returns '/node/32'
$nid = 32;
$options = ['absolute' => TRUE];
$url = Url::fromRoute('entity.node.canonical', ['node' => $nid], $options);
// make a string
$url_string = $url->toString();
```

Create a Node Alias

URL aliases are entities so you create them like you would any entity. Be sure to save() them.

```

$node_path = "/node/32";
$new_alias = "/test-node";

/** @var \Drupal\path_alias\PathAliasInterface $path_alias */

$my_node_alias = \Drupal::entityTypeManager()->getStorage('path_alias')->create([
  'path' => $node_path,
  'alias' => $new_alias,
  'langcode' => 'en',
]);
$my_node_alias->save();

```

Get the current Path

This returns the current relative path. For node pages, the return value will be in the form "/node/32" For taxonomy "taxonomy/term/5", for user "user/2" if it exists otherwise it will return the current request URI.

```

$currentPath = \Drupal::service('path.current')->getPath();

// Or with alias and query string.

$path = \Drupal::request()->getRequestUri();

// Or

$url_string = Url::fromRoute('<current>')->toString();

```

Get current nid, node type and title

There are two ways to retrieve the current node – via the request or via the route

```

$node = \Drupal::request()->attributes->get('node');

$nid = $node->id();

```

OR

```

$node = \Drupal::routeMatch()->getParameter('node');

if ($node instanceof \Drupal\node\NodeInterface) {
  // You can get nid and anything else you need from the node object.

  $nid = $node->id();
  $nodeType = $node->bundle();
  $nodeTitle = $node->getTitle();
}

```

If you need to use the node object in `hook_preprocess_page()` on the preview page, you need to use the "node_preview" parameter, instead of the "node" parameter:

```

function mymodule_preprocess_page(&$vars) {

  $route_name = \Drupal::routeMatch()->getRouteName();

  if ($route_name == 'entity.node.canonical') {
    $node = \Drupal::routeMatch()->getParameter('node');
  }
  elseif ($route_name == 'entity.node.preview') {
    $node = \Drupal::routeMatch()->getParameter('node_preview');
  }
}

```

And from <https://drupal.stackexchange.com/questions/145823/how-do-i-get-the-current-node-id> when you are using or creating a custom block then you have to follow this code to get current node id. Not sure if it is correct

```

use Drupal\Core\Cache\Cache;

$node = \Drupal::routeMatch()->getParameter('node');

if ($node instanceof \Drupal\node\NodeInterface) {
  $nid = $node->id();
}

// for cache
public function getCacheTags() {
  //With this when your node changes your block will rebuild
  if ($node = \Drupal::routeMatch()->getParameter('node')) {
    //if there is node add its cachetag
    return Cache::mergeTags(parent::getCacheTags(), ['node:' . $node->id()]);
  }
  else {
    //Return default tags instead.
    return parent::getCacheTags();
  }
}

public function getCacheContexts() {
  //if you depend on \Drupal::routeMatch()
  //you must set context of this block with 'route' context tag.
  //Every new route this block will rebuild
  return Cache::mergeContexts(parent::getCacheContexts(), ['route']);
}

```

How to get current Route name

A Drupal route is returned in the form of a string e.g. view.files_browser.page_1

```
$current_route = \Drupal::routeMatch()->getRouteName();
```

It returns "entity.node.canonical" for the nodes, "system.404" for the 404 pages, "entity.taxonomy_term.canonical" for the taxonomy pages, "entity.user.canonical" for the users and custom route name that we define in `modulename.routing.yml` file.

Get current Document root path

This will return the current document root path like "/var/www/html/project1".

```
$image_path = \Drupal::service('file_system')->realpath();
```

Retrieve URL argument parameters

You can extract the url arguments with

```
$current_path = \Drupal::service('path.current')->getPath();
$path_args = explode('/', $current_path);
$term_name = $path_args[3];
```

For <https://txg.ddev.site/newsroom/search/?country=1206>

Retrieve query and GET or POST parameters (\$_POST and \$_GET)

For get variables

```
$query = \Drupal::request()->query->get('name');
$name = $_GET['abc'];
```

For POST variables:

```
$name = \Drupal::request()->request->get('name');
//or
$name = $_POST['abc'];
```

For all items in a GET:

```
$query = \Drupal::request()->query->all();
$search_term = $query['query'];
$collection = $query['collection'];
```

Be wary about caching. From <https://drupal.stackexchange.com/questions/231953/get-in-drupal-8/231954#231954> the code provided only works the first time so it is important to add a '#cache' context in the markup.

```
namespace Drupal\newday\Controller;

use Drupal\Core\Controller\ControllerBase;

class NewdayController extends ControllerBase {

  public function new() {
    $day = [
      "#markup" => \Drupal::request()->query->get('id'),
    ];
    return $day;
  }
}
```

The request is being cached, you need to tell the system to vary by the query arguments:

```
$day = [
  "#markup" => \Drupal::request()->query->get('id'),
  '#cache' => [
    'contexts' => ['url.query_args:id'],
  ],
];
```

More about caching render arrays: <https://www.drupal.org/docs/8/api/render-api/cacheability-of-render-arrays>

Modify URL Aliases programmatically with hook_pathauto_alias_alter

The [pathauto](#) contrib module includes a nice hook that you can use to modify url aliases on the fly.

You just do your necessary checks (the current entity is stored in \context['data']) and change the alias that is passed. Pathauto does the rest.

As implemented in a module file.

```

/**
 * Implements hook_pathauto_alias_alter().
 *
 * Note. This function is a stopgap measure to handle pathauto
 * token failing to return the parent menu item alias.
 *
 * Using the pattern:
 * [node:menu-link:parent:url:path]/[node:title]
 * never returns the parent alias for some inexplicable reason.
 * This works fine on a fresh Drupal 9 site. Much research
 * yielded no results so this function hijacks pathauto's
 * efforts to create the alias. It checks for the presence
 * of the parent menu alias.
 *
 * That parent alias is inserted in the path if it isn't found.
 * If the parent alias is correctly inserted, it should not impact
 * the correct functioning of pathauto and token.
 */
function diri_pathauto_alias_alter(&$alias, array &$context) {

    // Change the alias if it doesn't include the parent item.
    /** @var Drupal\pathauto\Entity\PathautoPattern $pattern*/
    $pattern = $context["pattern"];
    /* The pattern will be something like this
     * [node:menu-link:parent:url:path]/[node:title]
     * so I could test for it
     * but it doesn't seem necessary.
     * Of course, if they change the pattern later this
     * pattern will overwrite it.
     */
    $bundles = ['page', 'audience', 'overview', 'program_area'];
    if (in_array($context["bundle"], $bundles)) {
        $op = $context['op'];
        if (in_array($op, ['insert', 'update', 'bulkupdate'])) {
            $parts = explode('/', $alias);
            if (count($parts) == 2) {
                if ($parts[0] == "") {
                    //Missing the parent link.
                    $id = $pattern->id();
                    //Is this the default pathauto pattern?
                    if ($id == "default") {
                        //Is there a parent link?
                        $nid = $context["data"]["node"]->id();
                        $parent_link = _findParentMenuItem($nid);
                        if ($parent_link) {
                            $parent_alias = $parent_link['#url']->toString();
                            //Update the alias to include the parent alias.
                            if ($parent_alias) {
                                $alias = $parent_alias . $alias;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Also see an example at <https://makedrupaleeasy.com/articles/drupal-version-7-9-how-update-alias-programmatically-using-value-field>

Drupal l() is deprecated

DEPRECATED

The l() method (lower case letter L) was a convenience wrapper for the link generator service's generate() method. So do this instead:

```
use Drupal\Core\Url;
use Drupal\Core\Link;

$url = Url::fromRoute('entity.node.edit_form', ['node' => $nid]);
$project_link = Link::fromTextAndUrl(t('Open Project'), $url);
$project_link = $project_link->toRenderable();
// If you need some attributes.
$project_link['#attributes'] = ['class' => ['button', 'button-action', 'button--primary', 'button--small']];
print render($project_link);
```

Reference links

- [Good reference from 2017 for creating links in Drupal](#)
- [#! code: Drupal 9: Programmatically Creating And Using URLs And Links, March 2022](#)

[Back to top](#)

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

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

Logging

TABLE OF CONTENTS

- [Quick log to watchdog](#)
- [Log an email notification was sent to the the email address for the site.](#)
- [Logging from a service using dependency injection](#)
- [Another example using the logging via dependency injection](#)
- [Logging exceptions from a try catch block](#)
- [Display a message in the notification area](#)
- [Display a variable while debugging](#)
- [Reference](#)

views 75

Quick log to watchdog

With the Database Logging (dblog) module enabled, you can easily log messages to the database log (watchdog table)

```
//Class with method name
$method = __METHOD__;

// Function name only.
$function = __FUNCTION__;

// Function name, filename, line number.
$str = __FUNCTION__." in __FILE__ at __LINE__;

\Drupal::logger('test')->info("method = $method");
\Drupal::logger('test')->info("Something goofed up at $str");
\Drupal::logger('test')->debug("Something goofed up at $str");
\Drupal::logger('test')->critical("Something goofed up at $str");

\Drupal::service('logger.factory')->get('test')->error('This is my error message');
```

The parameter “test” used above is typically the module name. It is stored in the “type” field.

You can call difference methods such as `info`, `warning` etc. which populate the `severity` field with an integer indicating the severity of the issue.

The methods are defined in `Drupal\Core\Logger\RfcLoggerTrait`:

- `emergency($message, $context)`
- `alert($message, $context)`
- `critical($message, $context)`
- `error($message, $context)`
- `warning($message, $context)`
- `notice($message, $context)`
- `info($message, $context)`
- `debug($message, $context)`

More at <https://www.drupal.org/docs/8/api/logging-api/overview>

Log an email notification was sent to the the email address for the site.

```
$email_config = \Drupal::config('system.site');
$to = $email_config->get('mail');
// Display message to screen.
$messenger->addMessage("sent a message to $to");
// Log it.
\Drupal::logger('DIR')->info("Email notification send to $to succeeded");
```

Incidentally, calling \Drupal::logger like this

```
\Drupal::logger('my_module')->error('This is my error message');
```

actually does this under the covers:

```
\Drupal::service('logger.factory')->get('hello_world')->error('This is my error message');
```

Logging from a service using dependency injection

From a controller e.g. WebsphereAddress.php

In the websphere_commerce.services.yml specify the @logger.factory to be passed into the constructor.

```
services:
  websphere_commerce.address:
    class: Drupal\websphere_commerce\WebSphereAddressService
    arguments: ['@config.factory', '@logger.factory']
```

In the WebsphereAddress.php file specify use statements:

```
use Drupal\Core\Logger\LoggerChannelFactory;
use Drupal\Core\Logger\LoggerChannelFactoryInterface;
```

Create a protected var to store the logger service:

```
/**
 * @var Drupal\Core\Logger\LoggerChannelFactory
 */
protected $logger;
```

Here is the constructor:

```
/**
 * WebsphereAddress constructor.
 *
 * @param \Drupal\Core\Config\ConfigFactoryInterface $config_factory
 *   Config factory.
 * @param \Drupal\Core\Logger\LoggerChannelFactoryInterface $channel_factory
 *   Logger factory.
 */
public function __construct(ConfigFactoryInterface $config_factory, LoggerChannelFactoryInterface $channel_factory) {
  $this->websphereConfig = $config_factory->get('websphere_commerce.api_settings');
}
```

Log errors.

```
if ($response['status'] == API_ERROR) {  
    $this->logger->get('websphere_commerce')->alert("Error saving Shipping info to Websphere.");  
}
```

Another example using the logging via dependency injection

From the excellent folks at symfonycasts.com who have a sweet [Drupal 8 course](#) which is still relevant and worth checking out.

In your `dino_roar.services.yml` file, add the listener and specify the arguments of `[@logger.factory]`

Note. You can find the factory info with Drupal console:

```
$ drupal debug:container | grep log
```

one of the results specifies the factory which you can use below:

```
logger.factory      Drupal\Core\Logger\LoggerChannelFactory
```

or with Drush and devel

```
$ drush dcs log  
  
- logger.dblog  
- logger.drupaltodrush  
- logger.factory
```

So `dino_roar.dino_listener` will pass the `logger.factory` service to your `DinoListener` class.

```
dino_roar.dino_listener:  
  class: Drupal\dino_roar\Jurassic\DinoListener  
  arguments: ['@logger.factory']  
  tags:  
    - {name: event_subscriber}
```

in your `DinoListener.php` specify a constructor argument of `LoggerChannelFactoryInterface` and store it.

```

namespace Drupal\dino_roar\Jurassic;

use Drupal\Core\Logger\LoggerChannelFactoryInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpKernel\KernelEvents;

class DinoListener implements EventSubscriberInterface {

 /**
 * @var \Drupal\Core\Logger\LoggerChannelFactoryInterface
 */
 private $loggerChannelFactory;

public function __construct(LoggerChannelFactoryInterface $loggerChannelFactory) {

$this->loggerChannelFactory = $loggerChannelFactory;
}

public function onKernelRequest(GetResponseEvent $event) {
$request = $event->getRequest();
$shouldRoar = $request->query->get('roar');

if ($shouldRoar) {
$this->loggerChannelFactory->get('default')
->debug("Roar Requested ROOOOAAAARRR!");
}
}

public static function getSubscribedEvents() {
return [
KernelEvents::REQUEST => 'onKernelRequest',
];
}
}

```

Logging exceptions from a try catch block

In this controller, the try block calls the test() method which throws an exception. The catch block catches the exception and logs the message (and for fun displays a message in the notification area also.)

```

public function build() {

  try {
    $this->test();
  }
  catch (\Exception $e) {
    watchdog_exception('nuts_connect', $e);
    $messenger->addMessage("No, I got caught!");
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];

  return $build;
}

function test() {
  throw new \Exception("blah", 7);
}

```

Display a message in the notification area

You can display a message with:

```

$messenger = \Drupal::messenger();
$messenger->addMessage("a message");
$messenger->addError("error message");

```

Or

```

\Drupal::messenger()->addError("migration failed");

\Drupal::messenger()->addMessage($message, $type, $repeat);

```

Use `$repeat = FALSE` to suppress duplicate messages.

Specify `MessengerInterface::TYPE_STATUS`, `MessengerInterface::TYPE_WARNING`, or `MessengerInterface::TYPE_ERROR` to indicate the severity.

Don't forget

```
use Drupal\Core\Messenger\MessengerInterface;
```

Note. `addMessage()` adds `class="messages messages--status"` to the div surrounding your message while `addError` adds `class="messages messages--status"`. Use these classes to format the message appropriately.

When you need to display a message in a form, use the `$this->messenger()` that is provided by the `Drupal\Core\Messenger\MessengerTrait`:

```
$this->messenger()->addStatus($this->t('Running in Destructive Mode - Changes ARE committed to the database!'));
```

e.g.

```
\Drupal::messenger()->addMessage('Program pending, please assign team and initialize.', MessengerInterface::TYPE_WARNING);
```

Display a variable while debugging

You can use `var_dump` and `print_r` but sometimes it is difficult to see where they display.

```
$is_front = \Drupal::service('path.matcher')->isFrontPage();
$is_front = $is_front == TRUE ? "YEP" : "NOPE";
$messenger->addMessage("is_front = $is_front");
var_dump($is_front);
print_r($is_front);
```

The screenshot shows a macOS desktop with several browser tabs open in the top bar:

- selwynp
- Actions
- d9book
- Basic wr
- Online P
- Another tab with a blue icon

The main window displays a Drupal site install page. The URL in the address bar is `ddev93.ddev.site/test/example`. The page has a blue header with the Drupal logo and navigation links: Content, Structure, Appearance, Extend, Configuration, and People.

The main content area features a large blue banner with the text "Drush Site-Install". Below it, there are two message boxes:

- A green box containing a checkmark icon, the text "a message", and the message "sent a message to admin@example.com".
- A red box containing a red X icon and the text "error message".

A red arrow points from the text "var_dump() and print_r()" in the green message box to the "is_front = NOPE" line in the code at the top of the page.

Reference

- How to Log Messages in Drupal 8 by Amber Matz of Drupalize.me Updated October 2015 <https://drupalize.me/blog/201510/how-log-messages-drupal-8>
- Logging API updated January 2023 <https://www.drupal.org/docs/8/api/logging-api/overview>
- Drupal APIs <https://www.drupal.org/docs/drupal-apis>

[Back to top](#)

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

Page last modified: Apr 13 2023.

[Edit this page on GitHub](#)



Menus

TABLE OF CONTENTS

- [Dynamically change menu items with hook_preprocess_menu](#)
- [Permanently update menu links in a hook_update using entityQuery](#)
- [Add menu items with hook_update](#)
- [Permanently modify or delete menu items with hook_update](#)
- [Peer up a menu to its parents to see if it is a child of a content type](#)
- [Find all the children of a menu](#)
- [Build a menu and all its children](#)
- [Create custom Twig extension for rendering a menu](#)
- [Active Trail](#)
- [Get a node's menu item and more](#)
- [Create menu items in your custom module](#)
- [Resources](#)

Views 100

Dynamically change menu items with hook_preprocess_menu

In the example below, we're changing the labels of items in the user menu. The labels are changed from "login" and "log out" to "log the flock in" and "log the flock out." This can be implemented in a theme file as it is here.

```
/**  
 * Implements hook_preprocess_menu().  
 */  
  
function pega_academy_theme_preprocess_menu(&$vars, $hook) {  
  if ($hook == 'menu__account') {  
    $items = $vars['items'];  
    foreach ($items as $key => $item) {  
      if ($key == 'user.page') {  
        $vars['items'][$key]['title'] = [  
          '#markup' => 'Log the <i>flock</i> in!',  
          '#allowed_tags' => ['i'],  
        ];  
      }  
      if ($key == 'user.logout') {  
        $vars['items'][$key]['title'] = [  
          '#markup' => 'Log the <i>flock</i> out!',  
          '#allowed_tags' => ['i'],  
        ];  
      }  
    }  
  }  
}
```

Permanently update menu links in a hook_update using entityQuery

To update menu item links, you can use the following code (from a .install file).

```

function pdq_academy_core_update_8002() {

$uids = \Drupal::entityQuery('menu_link_content')
->condition('menu_name', 'pdq-wide-utility')
->execute();

foreach($uids as $uid) {
$menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($uid);

$title = $menu_link->getTitle();
if ($title === 'Support') {
$menu_link->set('weight', 2);
$menu_link->set('expanded', TRUE);
// $menu_link->set('title', 'yomama');
$menu_link->set('link', 'https://www.google.com');
$menu_link->save();
}

}
}
}

```

Add menu items with hook_update

Menus are config entities while menu items are content entities. Here, a hook_update creates some menu items and adds them to an existing menu.

```

function pdq_archive_core_update_8001() {
$item = [
'1' => ['Training', 'https://abc.pdq.com/', -51, TRUE],
'2' => ['Support', 'https://abc.pdq.com/search/product-support', -50, TRUE],
];

foreach($item as $item) {
$menu_link = Drupal\menu_link_content\Entity\MenuLinkContent::create([
'title' => $item[0],
'link' => ['uri' => $item[1]],
'menu_name' => 'pdq-wide-utility',
'weight' => $item[3],
'expanded' => $item[4],
]);
$menu_link->save();
}
}

```

Permanently modify or delete menu items with hook_update

Below, we use hook_update to grab all the menu items in the menu called pdf-wide-utility, then loop thru them, delete some, and change the weight of some.

```

function pdq_archive_core_update_8001() {

  // Update and remove unused menu items
  $mids = \Drupal::entityQuery('menu_link_content')
    ->condition('menu_name', 'pdq-wide-utility')
    ->execute();

  foreach($mids as $mid) {

    $menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($mid);

    $title = $menu_link->getTitle();
    if ($title === 'Pdq.com' || $title === 'PDQ Community' || $title === 'Careers') {
      $menu_link->delete();
    }
    if ($title === 'Support') {
      $menu_link->set('weight',2);
      $menu_link->set('expanded', TRUE);
      //   $menu_link->set('title','yomama');
      //   $menu_link->set('link', 'https://www.google.com');
      $menu_link->save();
    }
  }
}

```

If you need to get the parent value so you can make a menu item a child, use:

```
$parent_id = $menu_link->getPluginId();
```

and

```
$menu_link->set('parent', $parent_id);
```

Peer up a menu to its parents to see if it is a child of a content type

Here, we need to display a sidebar if the current node is both a page and a child (or any level of offspring, e.g., grandchild, great-grandchild, etc.) of a content type "unit." This means we need to "peer" up the menu chain to see what kind of parent the node has.

This example was implemented in a .theme file.

When hook preprocess_node is called for my content type, and we are viewing a full node, we grab the nid and call the _check_ancestry_for_unit(), which looks up the menu chain for a menu item that points to a node of type "unit". If there is one, we display the sidebar, i.e., we set the \$variables ['show_sidebar_menu'] to TRUE.

```

function txg_preprocess_node(&$variables) {
  $node = $variables['node'];

  if (($node->getType() == 'page') && ($view_mode == 'full')) {
    // This will only be true if it is a node route.
    if ($node = \Drupal::request()->attributes->get('node')) {
      if (is_string($node)) {
        $node_id = $node;
      }
      else {
        $node_id = $node->id();
      }
      $show_sidebar = _check_ancestry_for_unit($node_id);
      $variables['show_sidebar_menu'] = $show_sidebar;
    }
  }
}

```

Here is the function _check_ancestry_for_unit():

```

/**
 * Go look up the menu chain for a unit.
 *
 * @param int $node_id
 *   Start with this node.
 * @param int $unit_nid
 *   Fill in the unit nid that you found.
 *
 * @return bool
 *   Return TRUE if there is a unit in the lineage.
 *
 * @throws \Drupal\Component\Plugin\Exception\PluginException
 */
function _check_ancestry_for_unit(int $node_id, &$unit_nid = 0, &$menu_item_title = '') {
  /** @var \Drupal\Core\Menu\MenuLinkManagerInterface $menu_link_manager */
  $menu_link_manager = \Drupal::service('plugin.manager.menu.link');
  $links = $menu_link_manager->loadLinksByRoute('entity.node.canonical', ['node' => $node_id]);

  /** @var \Drupal\Core\Menu\MenuLinkContent $link */
  $link = reset($links);
  if (!empty($link)) {
    while ($parent_menu_id = $link->getParent()) {

      // Create a menu item.
      /** @var \Drupal\Core\Menu\MenuLinkInterface $parent */
      $parent_menu_item = $menu_link_manager->createInstance($parent_menu_id);
      $parent_url = $parent_menu_item->getUrlObject();

      // Is the parent an External link e.g. nytimes.com?
      $external = $parent_url->isExternal();
      if ($external) {
        return FALSE;
      }

      // Internal links (could be <nolink>)
      $internal = $parent_url->isRouted();
      $parent_route = $parent_url->getRouteParameters();
      // When $parent_route is <nolink>.
      if (empty($parent_route)) {
        return FALSE;
      }

      // Parent is an internal link.
      // Does parent menu item refer to a unit?
      if (isset($parent_route['node'])) {
        $parent_nid = $parent_route['node'];
        $parent_node = Node::load($parent_nid);
        $parent_type = $parent_node->getType();
        if ($parent_type == 'unit') {
          $unit_nid = $parent_nid;
          $menu_item_title = $parent_menu_item->getTitle();
          return TRUE;
        }
        $links = $menu_link_manager->loadLinksByRoute('entity.node.canonical', ['node' => $parent_nid]);
        $link = reset($links);
      }
      else {
        // Some other kind of internal link.
        return FALSE;
      }
    }
  }
  return FALSE;
}

```

Find all the children of a menu

From a .module file, I needed to load a dropdown with items from the main menu.

You can load the menu up with this:

```
use Drupal\Core\Menu\MenuTreeParameters;

function get_menu_tree($menu_name) {
  $parameters = new MenuTreeParameters();

  // Only enabled items.
  $parameters->onlyEnabledLinks();

  // Load the tree.
  $tree = \Drupal::menuTree()->load($menu_name, $parameters);
  return $tree;
}
```

This is basically the same as:

```
$sub_nav = $menu_tree->load('main', new \Drupal\Core\Menu\MenuTreeParameters());
```

And here we call `get_menu_tree()` and then loop through the menu items to build some arrays to be rendered via a twig template. We loop through the top level, skipping non-nodes:

```

function get_offices() {

    //Get drupal main menu
    $tree = get_menu_tree('main');
    $menu_tree = \Drupal::menuTree();

    // Build a renderable array from the tree which fills in all the children in
    // item[below]
    $menu_render_array = $menu_tree->build($tree);
    $storage = [];
    $nid = 0;
    foreach ($menu_render_array['#items'] as $item) {
        $url = $item['url'];
        $route = $url->getRouteParameters();
        // Skip menu items that aren't nodes.
        if (!isset($route['node'])) {
            continue;
        }
        $nid = $route['node'];
        $node = Node::load($nid);
        $type = $node->getType();
        if ($type == "area_of_focus") {
            $storage[] = [
                'title' => $item['title'],
                'value' => $nid
            ];
        }
    }

    // Process children:
    foreach ($item['below'] as $child) {
        $url = $child['url'];
        $route = $url->getRouteParameters();
        $nid = 0;
        $type = "";
        if (isset($route['node'])) {
            $nid = $route['node'];
            $node = Node::load($nid);
            $type = $node->getType();
        }
        // Only add units.
        if ($type == 'unit') {
            $storage[] = [
                'title' => '--' . $child['title'] . ' (' . $type . ')',
                'value' => $nid
            ];
        }
    }
    return $storage;
}

```

And the template that is used to display the dropdown, node--news-stories-landing-page.html.twig:

```

<form action="#" class="filter-form">
<div class="search-form">
<input type="search" placeholder="Search Events by Keyword" title="Type search text here">
<button type="button"><i class="icon icon-search"></i><span class="show-for-sr">Search</span></button>
</div>
<div class="grid-x grid-margin-x align-justify">

</div>
</form>

```

And the form looks like this:

Search Events by Keyword



By Office

Select Office

By Topic

Select Topic

By Continent

Select Continent

By Country

Select Country

Build a menu and all its children

This also looks pretty interesting, but I haven't tried it. It is from <https://stackoverflow.com/questions/54245942/drupal-8-menulinkcontent-get-all-children-via-loadbyproperties/54254491>

```
function generateSubMenuTree(&$output, &$input, $parent = FALSE) {
  $input = array_values($input);
  foreach ($input as $key => $item) {
    //If menu element disabled skip this branch
    if ($item->link->isEnabled()) {
      $key = 'submenu-' . $key;
      $name = $item->link->getTitle();
      $url = $item->link->getUrlObject();
      $url_string = $url->toString();

      //If not root element, add as child
      if ($parent === FALSE) {
        $output[$key] = [
          'name' => $name,
          'tid' => $key,
          'url_str' => $url_string
        ];
      }
      else {
        $parent = 'submenu-' . $parent;
        $output['child'][$key] = [
          'name' => $name,
          'tid' => $key,
          'url_str' => $url_string
        ];
      }
    }

    if ($item->hasChildren) {
      if ($item->depth == 1) {
        generateSubMenuTree($output[$key], $item->subtree, $key);
      }
      else {
        generateSubMenuTree($output['child'][$key], $item->subtree, $key);
      }
    }
  }
}
```

It is called with:

```
//Get drupal menu
$sub_nav = \Drupal::menuTree()->load('main', new \Drupal\Core\Menu\MenuTreeParameters());

//Generate array
generateSubMenuTree($menu_tree2, $sub_nav);
```

Create custom Twig extension for rendering a menu

Note. The module [twig_tweak module](#) can do all this with one line of code:

```
{{ drupal_menu('main', 2, 3, TRUE) }}
```

More at: <https://www.drupal.org/docs/8/modules/twig-tweak/cheat-sheet>

From <https://www.drupal.org/forum/support/theme-development/2015-01-29/rendering-a-menu-in-twig-drupal-8>, Peter from Dusseldorf shows how to render a menu into a render array. He runs through the whole load, transform (with manipulators) and build. He does this through the magic of a twig extension. So, in modules/custom/custom_module/src/Twig/RenderMenuExtension.php:

```
namespace Drupal\custom_module\Twig;

class RenderMenuExtension extends \Twig_Extension {

    /**
     * @return array
     */
    public function getFunctions() {
        return [
            new \Twig_SimpleFunction('renderMenu', [$this, 'renderMenu']),
        ];
    }

    /**
     * Provides function to programmatically rendering a menu
     *
     * @param String $menu_name
     *   The machine configuration id of the menu to render
     */
    public function renderMenu(string $menu_name) {
        $menu_tree = \Drupal::menuTree();

        // Build the typical default set of menu tree parameters.
        $parameters = $menu_tree->getCurrentRouteMenuTreeParameters($menu_name);

        // Load the tree based on this set of parameters.
        $tree = $menu_tree->load($menu_name, $parameters);

        // Transform the tree using the manipulators you want.
        $manipulators = [
            // Only show links that are accessible for the current user.
            ['callable' => 'menu.default_tree_manipulators:checkAccess'],
            // Use the default sorting of menu links.
            ['callable' => 'menu.default_tree_manipulators:generateIndexAndSort'],
        ];
        $tree = $menu_tree->transform($tree, $manipulators);

        // Finally, build a renderable array from the transformed tree.
        $menu = $menu_tree->build($tree);

        $menu['#attributes']['class'] = 'menu ' . $menu_name;

        return ['#markup' => drupal_render($menu)];
    }
}
```

Oh, and you do need to implement the getFunctions() in your class. It looks something like this:

```
public function getFunctions() {
    $context_options = ['needs_context' => TRUE];
    $all_options = ['needs_environment' => TRUE, 'needs_context' => TRUE];
    return [
        new \Twig_SimpleFunction('drupal_render', [$this, 'renderMenu']),
    ];
}
```

In custom_module.services.yml:

```

services:
  custom_module.render_menu_extension:
    class: Drupal\custom_module\Twig\RenderMenuExtension
    tags:
      - { name: twig.extension }

```

To render your menu in the template via this twig function call:

```
 {{ renderMenu('main') }}
```

Active Trail

From

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Menu%21MenuTreeParameters.php/property/MenuTreeParameters%3A%3AactiveTrail/9.3.x>

The IDs from the currently active menu link to the root of the whole tree.

Active trail is an array of menu link plugin IDs, representing the trail from the currently active menu link to the ("real") root of that menu link's menu. This does not affect the way the tree is built. It is only used to set the value of the `inActiveTrail` property for each tree element.

In the code below, I grab the active trail for an item that is in a menu. Then, I grab all the links for the current `node_id`, pull off the first one, grab the plugin (which is `menu_link_content:957297e4-38eb-4502-868a-668407c71a44` – the id from the `menu_tree` table), and get the parameters (all the juicy goodness about this menu item). You can find a nice trail back up the menu chain along the `active_trail`. See `activeTrail` in the debug variable dump below.

```

// Get current item's menu
/** @var \Drupal\Core\Menu\MenuLinkManagerInterface $menu_link_manager */
$menu_link_manager = \Drupal::service('plugin.manager.menu.link');
$links = $menu_link_manager->loadLinksByRoute('entity.node.canonical', ['node' => $node_id]);
/** @var \Drupal\Core\Menu\MenuLinkContent $currentMenuItem */
$currentMenuItem = reset($links);

// grab plugin for the current item
$pluginId = $currentMenuItem->getPluginId();

/** @var \Drupal\Core\Menu\MenuLinkTreeInterface $menu_tree */
$menu_tree = \Drupal::service('menu.link_tree');
$parameters = $menu_tree->getCurrentRouteMenuTreeParameters($menu_name);

// grab the active trail.
$active_trail = array_keys($parameters->activeTrail);

```

```

$parameters = {Drupal\Core\Menu\MenuTreeParameters} [6]
  01 root = ""
  01 minDepth = null
  01 maxDepth = null
  ▾ 1 expandedParents = {array} [6]
    01 menu_link_content:957297e4-38eb-4502-868a-668407c71a44 = "menu_link_content:957297e4-38eb-4502-868a-668407c71a44"
    01 menu_link_content:8e5abff7-0e94-42be-993f-2d1b3d538071 = "menu_link_content:8e5abff7-0e94-42be-993f-2d1b3d538071"
    01 menu_link_content:ea010649-7a2a-40c3-8673-687a5fc04878 = "menu_link_content:ea010649-7a2a-40c3-8673-687a5fc04878"
    01 menu_link_content:ba000658-e485-41b5-b2eb-eaa10cf1f713 = "menu_link_content:ba000658-e485-41b5-b2eb-eaa10cf1f713"
    01 menu_link_content:d1b2acfb-7438-4be8-8a61-63a3bc090d03 = "menu_link_content:d1b2acfb-7438-4be8-8a61-63a3bc090d03"
    01 = ""
  ▾ 2 activeTrail = {array} [6]
    01 menu_link_content:957297e4-38eb-4502-868a-668407c71a44 = "menu_link_content:957297e4-38eb-4502-868a-668407c71a44"
    01 menu_link_content:8e5abff7-0e94-42be-993f-2d1b3d538071 = "menu_link_content:8e5abff7-0e94-42be-993f-2d1b3d538071"
    01 menu_link_content:ea010649-7a2a-40c3-8673-687a5fc04878 = "menu_link_content:ea010649-7a2a-40c3-8673-687a5fc04878"
    01 menu_link_content:ba000658-e485-41b5-b2eb-eaa10cf1f713 = "menu_link_content:ba000658-e485-41b5-b2eb-eaa10cf1f713"
    01 menu_link_content:d1b2acfb-7438-4be8-8a61-63a3bc090d03 = "menu_link_content:d1b2acfb-7438-4be8-8a61-63a3bc090d03"
    01 = ""
  ▾ 3 conditions = {array} [1]
    01 enabled = {int} 1

```

Extracting out the active trail gives this useful information:

```

▼ 1 $active_trail = {array} [6]
  0 = "menu_link_content:957297e4-38eb-4502-868a-668407c71a44"
  1 = "menu_link_content:8e5abbf7-0e94-42be-993f-2d1b3d538071"
  2 = "menu_link_content:ea010649-7a2a-40c3-8673-687a5fc04878"
  3 = "menu_link_content:ba000658-e485-41b5-b2eb-eaa10cf1f713"
  4 = "menu_link_content:d1b2acfb-7438-4be8-8a61-63a3bc090d03"
  5 =

```

Get a node's menu item and more

Here we get the current route's menu item using its nid then pull the link from the array using reset, and we can extract the URL as well as other exciting things. Mostly we want to check its children, parents etc. In the code below, we grab its URL as well as its parent and its title.

```

/** @var \Drupal\Core\Menu\MenuLinkManagerInterface $menu_link_manager */

$menu_link_manager = \Drupal::service('plugin.manager.menu.link');

$links = $menu_link_manager->loadLinksByRoute('entity.node.canonical', ['node' => $nid]);
$link = reset($links);
$urlobject = $link->getUrlObject();
$url_string = $urlobject->toString();
$x = $link->getParent(); //get the parent menu item GUID.
$y = $link->getTitle(); //get the title of the menu item.

$menu_name = $link->getMenuName(); //get the menu name e.g. "main"

```

Create menu items in your custom module

When creating a menu for your module, you need a YAML file like this from dev1 pageexample. The names (e.g. pageexample.description and pageexample.simple) are arbitrary, the title is the menu text, and the route_name comes from the routing yml file (web/modules/custom/pageexample/pageexample.routing.yml). The parent value is interesting.

```

pageexample_description:
  title: 'Page Example'
  route_name: pageexample_description
  parent: system.admin_reports
pageexample.simple:
  title: 'Simple page example - no arguments'
  route_name: pageexample_simple
  parent: system.admin_reports

```

The parent values are defined in other *.links.menu.yml files and especially the Structure link, which is defined in the core System module in system.links.menu.yml. Here we are adding a link to appear under the Reports menu of Drupal.

To create menu links programmatically, see <https://drupal.stackexchange.com/questions/197073/how-do-i-create-menu-links-programmatically/197076#197076>

To edit menu links programmatically, see <https://drupal.stackexchange.com/questions/235516/how-do-i-programmatically-update-or-delete-menu-items>

First you will have to load the entity. Either way works:

```
$menu_link = MenuLinkContent::load($menu_link_id);
```

Or ...

```
$menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($menu_link_id);
```

Next you can update value using set() method or through the magic method __set...

```
$menu_link->expanded = TRUE;
```

To save, simply call the save() method. To delete, call the delete() method.

Resources

- To create menu links programmatically see <https://drupal.stackexchange.com/questions/197073/how-do-i-create-menu-links-programmatically/197076#197076>
 - To edit menu links programmatically, see <https://drupal.stackexchange.com/questions/235516/how-do-i-programmatically-update-or-delete-menu-items>
 - Active Trail
<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Menu%21MenuTreeParameters.php/property/MenuTreeParameters%3A%3AactiveTrail/9.3.x>
 - The [Twig Tweak module](#) can do some great menu magic. Cheat sheet at <https://www.drupal.org/docs/8/modules/twig-tweak/cheat-sheet>
 - [#! code: Drupal 9: Creating A Category Menu Using Derivers, August 2022.](#)
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) 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.

Migration

TABLE OF CONTENTS

- [Import content from another Migration into Paragraphs](#)
- [Resources](#)

views 154

Import content from another Migration into Paragraphs

This migration Process snippet example demonstrates how to populate the fields of a Drupal Paragraph. Whereas standard fields can be simply mapped 1:1, and its value attribute (value) is implied (derived), a Paragraph requires *both* a target_id and target_revision_id. Annotations included inline to demonstrate what is going on within.

```
# Field name is `field_paragraph_authors`, specific property `target_id`:  
field_paragraph_authors/target_id:  
- plugin: migration_lookup  
  # Dependent Migration called `migration_paragraph_linked_author`  
  migration: migration_paragraph_linked_author  
  # Don't create stub content if the row currently being processes does not map to an item in the earlier-run Migration  
  no_stub: true  
  # How to map this Migration with the earlier-run Migration  
  source: sku  
- plugin: skip_on_empty  
  # Method: If empty, skip only this field mapping ('process'), not the entire Row ('row')  
  method: process  
- plugin: extract  
  # This destination property is 1st element in the migration-lookup array.  
  index:  
    - 0  
# Other half of `field_paragraph_authors`, specific property `target_revision_id`:  
field_paragraph_authors/target_revision_id:  
- plugin: migration_lookup  
  migration: migration_paragraph_linked_author  
  no_stub: true  
  source: sku  
- plugin: skip_on_empty  
  method: process  
- plugin: extract  
  # This destination property is the 2nd element in the migration-lookup array.  
  index:  
    - 1
```

Resources

- 31 days of Drupal migrations by Mauricio Dinarte August 2019 <https://understanddrupal.com/migrations>
- Stop waiting for Feeds module: how to import RSS in Drupal 8 by Campbell Vertesi June 2017 <https://ohthehugemanatee.org/blog/2017/06/07/stop-waiting-for-feeds-module-how-to-import-remote-feeds-in-drupal-8>
- Issue on Drupal.org where Mike Ryan, the author of the migrate module, addresses how to start a migration programmatically <https://www.drupal.org/project/drupal/issues/2764287>
- Video from Twin Cities Drupal community where Mauricio Dinarte and Benjamin Melançon demonstrate how to implement migrations - June 2018. <https://www.youtube.com/watch?v=eBP2vQlwx-o>
- Migrating to Drupal From Alternate Sources by Joshua Turton showing migrations from CSV, XML/RSS and Wordpress into Drupal 8 - November 2018 <https://www.phase2technology.com/blog/migrating-drupal-alternate-sources>

- Nice intro to Drupal 8 migration by Chris of Redfin Solutions from Nov 2017. <https://redfinsolutions.com/blog/understanding-drupal-8-migrate-api>
 - Migrating files and images on Drupal.org updated Feb 2023. <https://www.drupal.org/docs/drupal-apis/migrate-api/migrate-destination-plugins-examples/migrating-files-and-images>
 - Drupal 8 content migrations from CSV or spreadsheet August 2020 <https://atendesigngroup.com/articles/drupal-8-content-migrations-csv-spreadsheet>
-
-

[Back to top](#)

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

Page last modified: Apr 28 2023

[Edit this page on GitHub](#)

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

Modal Dialogs

TABLE OF CONTENTS

- [Overview](#)
- [Dialog title](#)
- [Links to slide-in dialogs](#)
- [Modal dialog example](#)
 - [Passing entities as parameters](#)
- [Modal form example](#)
- [Slide-in dialog/Off-canvas dialog](#)
- [Slide-in Dialog Example](#)
- [Block with a link to popup a custom modal dialog](#)
- [No-code modal dialogs](#)
- [Resources](#)

views 98

Overview

There are several ways to create modal dialogs in Drupal. Modal dialogs provide a way to display additional information without having to reload the entire page. These dialogs can display anything you can imagine including static text, an image, a node, a form, a view or any custom markup.

Dialog title

You define the title for the dialog by specifying it in the *.routing.yml file. In the file below the title is in the `_title` key:

```
taa.srp_delete_citation:  
  path: '/taa/admin/citation/delete/citation/{citation}'  
  defaults:  
    _title: 'Delete Citation'  
    _form: 'Drupal\taa\Form>DeleteCitationForm'
```

The _title appears here

Incomplete Publisher Citation 225906

[Add New Citation](#)

[edit citation \(admin only\)](#)

[delete citation \(admin only\)](#)

Publisher Citation

Comment Title: Openly Accessible Citations

Links to slide-in dialogs

In your controller (or in a form) you can define links in your render array which can fire off the slide-in dialogs. E.g.

```
use Drupal\Component\Serialization\Json;

$submit_for_review_link = [
  '#type' => 'link',
  '#title' => t('Submit for Review'),
  '#url' => Url::fromRoute('taa_teks_publisher.submit_for_review', ['node' => $node->id(), 'return_type' => 'program']),
  '#attributes' => [
    'class' => ['use-ajax', 'button'],
    'id' => 'edit-program-' . $node->id(),
    'data-dialog-renderer' => 'off_canvas',
    'data-dialog-type' => 'dialog',
    'data-dialog-options' => Json::encode([
      [
        'width' => '600',
      ],
    ]),
  ],
];
```

Notice the data-dialog-options. 'off_canvas' is the magic sauce. Also note that the parameters that are passed to the form are 'node' and 'return_type' which are specified in the route below also.

Note the class 'button' will make this link show up as a button. If you remove it, this will instead just be a link.

You could easily make this a modal dialog by specifying:

```
'data-dialog-type' => 'modal',
'data-dialog-options' => Json::encode([ 'width' => 'auto',]),
```

Modal dialog example

This example displays a page with links to pop up two different modal dialogs. Clicking on either of the links will display a slightly different modal dialogs. The page looks like this:

Modal Examples (example1)



Route: `modal_examples.example1`

[Link to a custom modal \(modal1\)](#)
[Link to a node in a dialog \(modal2\)](#)

Here is the routing file at `web/modules/custom/modal_examples/modal_examples.routing.yml`.

The first route (`modal_examples.example1`) displays the links. The second route (`modal_examples.modal1`) allows the modal be displayed on a page on its own. This route also has parameters which can be useful when you want to display some relevant variable information in the dialog.

```
# Controller with buttons to open modals
modal_examples.example1:
  path: '/modal-examples/example1'
  defaults:
    _title: 'Modal Examples (example1)'
    _controller: '\Drupal\modal_examples\Controller\ModalExamplesController::buildExample1'
  requirements:
    _permission: 'access content'

# first modal
modal_examples.modal1:
  path: '/modal-examples/modal1/{program_id}/{type}'
  defaults:
    _title: 'Modal 1 with parameters'
    _controller: '\Drupal\modal_examples\Controller\ModalExamplesController::buildModal1'
  requirements:
    _permission: 'access content'

  options:
    parameters:
      program_id:
      type:
    no_cache: 'TRUE'

# Second modal
# You can't use this route directly
# route for modal to display a teaser of a node in a dialog
modal_examples.modal2:
  path: '/modal-examples/create-my-dialog'
  defaults:
    _controller: '\Drupal\modal_examples\Controller\ModalExamplesController2::createDialogFromNode'
  requirements:
    _access: 'TRUE'
```

In `web/modules/custom/modal_examples/src/Controller/ModalExamplesController.php` are the functions for both the page (which displays the link) and first modal.

Add the parameters to the end of the `Url::fromRoute()` call to pass them to the modal. You will probably do this a lot.

```
#url' => Url::fromRoute('modal_examples.modal1', [
  'program_id' => 123,
  'type' => 'all',
]),
```

```
<?php
```

```
namespace Drupal\modal_examples\Controller;

use Drupal\Component\Serialization\Json;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Url;

/**
 * Returns responses for Modal Examples routes.
 */
class ModalExamplesController extends ControllerBase {

  public function buildExample1() {

    $route_name = \Drupal::routeMatch()->getRouteName();
    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t('Route: %route', ['%route' => $route_name]),
    ];

    $build['link-to-modal1'] = [
      '#type' => 'link',
      '#title' => t('Link to a custom modal (modal1)'),
      '#url' => Url::fromRoute('modal_examples.modal1', [
        'program_id' => 123,
        'type' => 'all',
      ]),
      '#attributes' => [
        'id' => 'view-correlation-' . 12345,
        'class' => ['use-ajax'],
        'aria-label' => 'View useful information pertaining to item ' . '12345',
        '#prefix' => '<div class="abcdef">',
        '#suffix' => '</div>',
        'data-dialog-type' => 'modal',
        'data-dialog-options' => Json::encode([
          [
            'width' => 'auto',
          ],
        ]),
      ],
    ];
  }

  $build['link-to-modal2'] = [
    '#type' => 'link',
    '#url' => new Url('modal_examples.modal2'),
    '#title' => 'Link to a node in a dialog (modal2)',
    '#prefix' => '<div class="pqrsst">',
    '#suffix' => '</div>',
    '#attributes' => [
      'class' => ['use-ajax'],
    ],
  ];
}
```

```

};

$build['#attached']['library'][] = 'core/drupal.dialog.ajax';

return $build;
}

public function buildModal1(int $program_id, string $type) {

$build['content'] = [
'#type' => 'item',
'#markup' => $this->t('Some useful information!'),
];
$build['other_content'] = [
'#type' => 'item',
'#markup' => $this->t('Program id: @program_id. Type: @type', [
'@program_id' => $program_id,
'@type' => $type,
]),
];
}

return $build;
}
}

```

The second modal is in ModalExamplesController2.php. This dialog uses the example from an example at <https://www.hashbangcode.com/article/drupal-9-creating-ajax-dialogs> which displays a teaser view of a node:

```

<?php

/**
 * @file
 *
 * Contains \Drupal\modal_examples\Controller\ModalExamplesController2.
 */
namespace Drupal\modal_examples\Controller;

use Drupal\Core\Ajax\OpenDialogCommand;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\OpenModalDialogCommand;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Form\FormBuilder;

/**
 * ModalExamplesController2 class.
 */
class ModalExamplesController2 extends ControllerBase {

/**
 * The form builder.
 *
 * @var \Drupal\Core\Form\FormBuilderInterface
 */
protected $formBuilder;

```

```

/**
 * The entity type manager.
 *
 * @var \Drupal\Core\Entity\EntityTypeManagerInterface
 */
protected $entityTypeManager;

/**
 * The ModalFormExampleController constructor.
 *
 * @param \Drupal\Core\Form\FormBuilderInterface $formBuilder
 *   The form builder.
 * @param \Drupal\Core\Entity\EntityTypeManagerInterface $entity_type_manager
 *   The entity type manager.
 */
public function __construct(FormBuilderInterface $formBuilder, EntityTypeManagerInterface $entity_type_manager) {
  $this->formBuilder = $formBuilder;
  $this->entityTypeManager = $entity_type_manager;
}

/**
 * {@inheritDoc}
 *
 * @param \Symfony\Component\DependencyInjection\ContainerInterface $container
 *   The Drupal service container.
 *
 * @return static
 */
public static function create(ContainerInterface $container) {
  return new static(
    $container->get('form_builder'),
    $container->get('entity_type.manager'),
  );
}

/*
 * Phil's open a dialog to display a node teaser.
 */
public function createDialogFromNode() {
  // Load a specific node.
  $node = $this->entityTypeManager->getStorage('node')->load(34);

  // Convert node into a render array.
  $viewBuilder = $this->entityTypeManager->getViewBuilder('node');
  $content = $viewBuilder->view($node, 'teaser');

  // Get the title of the node.
  $title = $node->getTitle();

  // Create the AjaxResponse object.
  $response = new AjaxResponse();

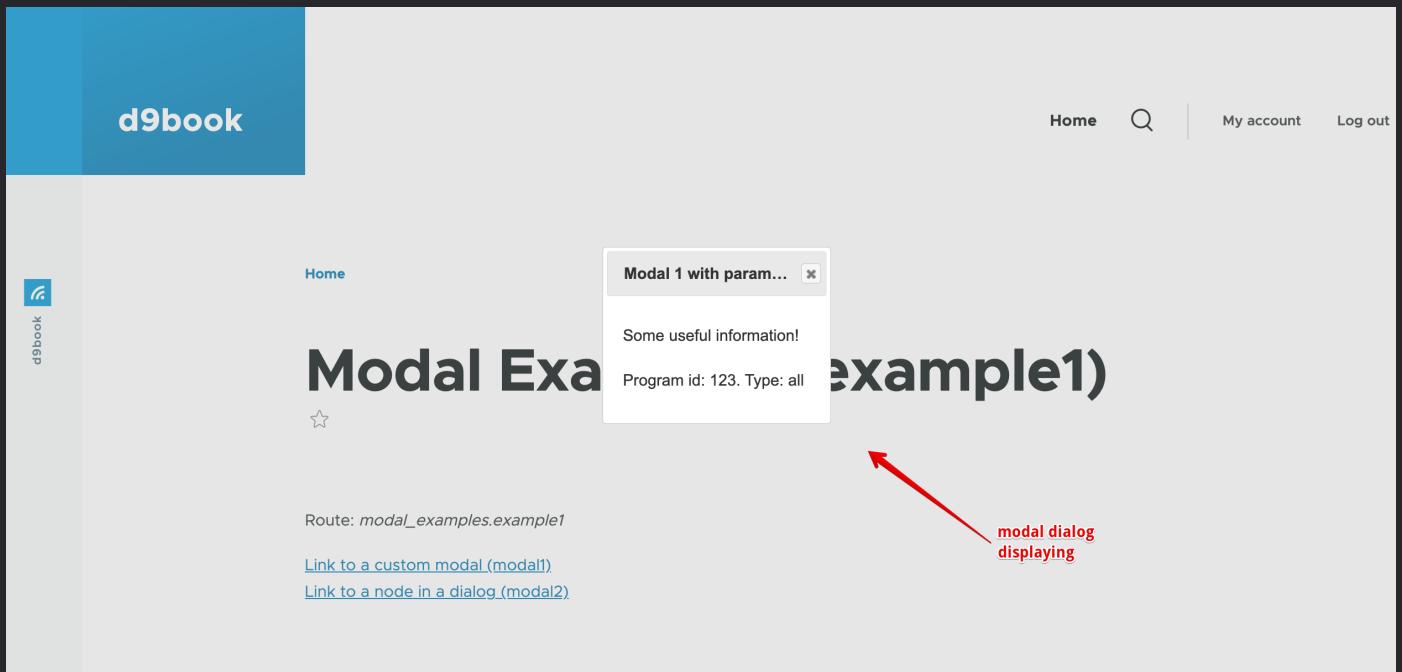
  // Attach the library needed to use the OpenDialogCommand response.
  $attachments['library'][] = 'core/drupal.dialog.ajax';
  $response->setAttachments($attachments);

  // Add the open dialog command to the ajax response.
  $response->addCommand(new OpenDialogCommand('#my-dialog-selector', $title, $content, ['width' => '70%']));
  return $response;
}

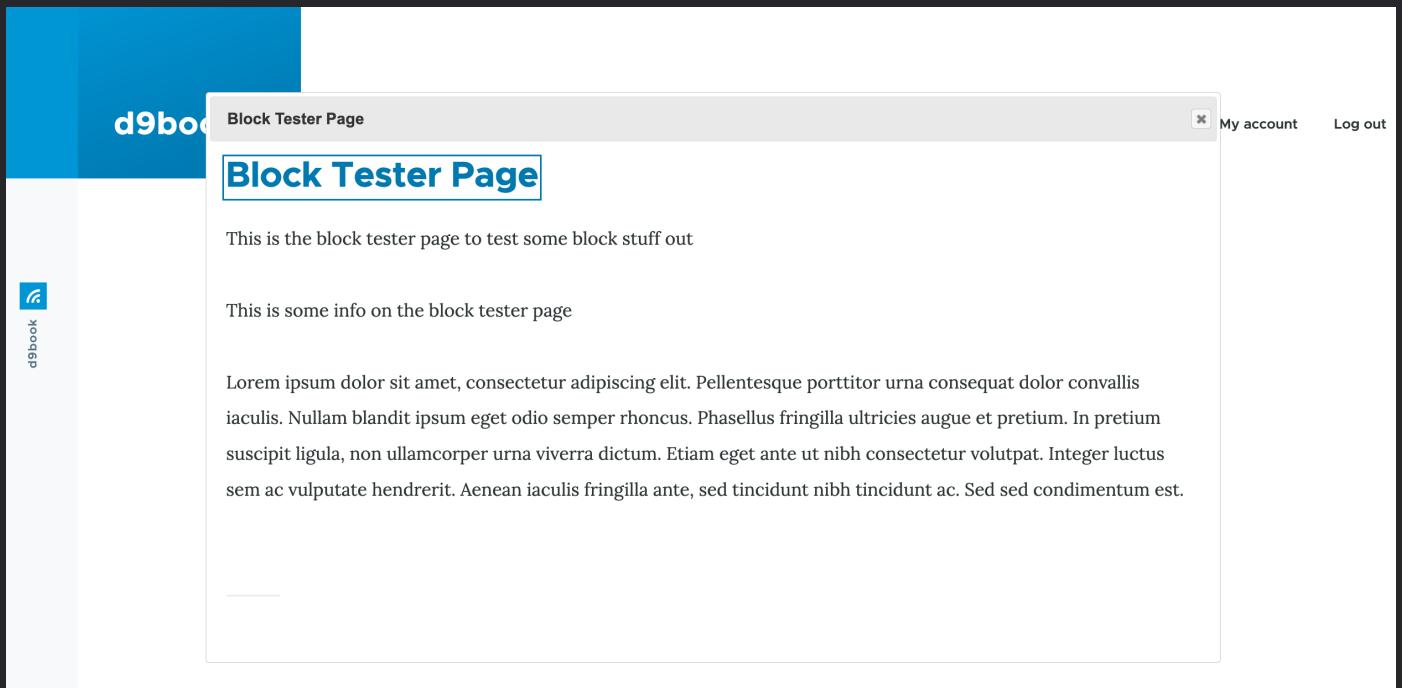
```

```
    }  
  
}
```

Here is the first modal dialog as it appears on the site when the modal1 link is clicked:



And the second



Passing entities as parameters

Sometimes it is useful to pass a node as a parameter so you can quickly grab fields from it. To do this, specify type: entity: node (or other type of entity) in the *.routing.yml file. See the example below where parameters: program and correlation are identified as nodes. In your code, you pass the node id (not the entire node) and Drupal will automatically load the node for you and pass it to the controller:

```

taa.view_votes:
path: '/srp/program/{program}/vote-view/correlation/{correlation}/{type}'
defaults:
_controller: '\Drupal\taa\Controller\ViewVotesController::content'
_title: 'Votes for Citation'
requirements:
_permission: 'view any saa item+view own saa item'
options:
parameters:
program:
type: entity:node
correlation:
type: entity:node
no_cache: 'TRUE'

```

The controller will look like this. Notice how we can call entity functions on the \$program and \$correlation entities.

```

public function content(EntityInterface $program, EntityInterface $correlation, string $type = 'all'): array {
$program_nid = $program->id();
$correlation_nid = $correlation->id();
$vote_number = $program->get('field_srp_vote_number')->value;

```

Modal form example

This example displays a form with a button. When a user presses the button, a dialog containing another form appears.

This code is a little convoluted as the link you click to open the modal form calls a controller function, `ModalExamplesController2::openModalForm()`, which loads the form and opens it with an ajax command, `OpenModalDialogCommand`. You can directly point a link to a form route (as you can find in other examples below.)

Also notice that controller uses dependency injection to load the form builder (which *is* the recommended way to do things.).

In `web/modules/custom/modal_examples/modal_examples.routing.yml` we first set up a route for the form which will be initially displayed. This is identified as `modal_examples.form1`.

Secondly, we set up a route for the popup form, `modal_examples.modal_form`.

```

modal_examples.form1:
path: '/modal-examples/form1'
defaults:
_form: 'Drupal\modal_examples\Form\ExampleForm'
_title: 'Form with a button to popup a modal form'
requirements:
_permission: 'administer site configuration'

modal_examples.modal_form:
path: '/modal-examples/modal_form'
defaults:
_title: 'The Modal Form'
_controller: '\Drupal\modal_examples\Controller\ModalExamplesController2::openModalForm'
requirements:
_permission: 'administer site configuration'
options:
_admin_route: TRUE

```

When we navigate to `/modal-example/form`, we see the following form. If we click the button labelled, "Click to see the Modal Form", a modal dialog will appear with a form in it.

Here is the initial form:

[Home](#)

Form with a button to popup a modal form

Please fill out the form below and click the button for more info.

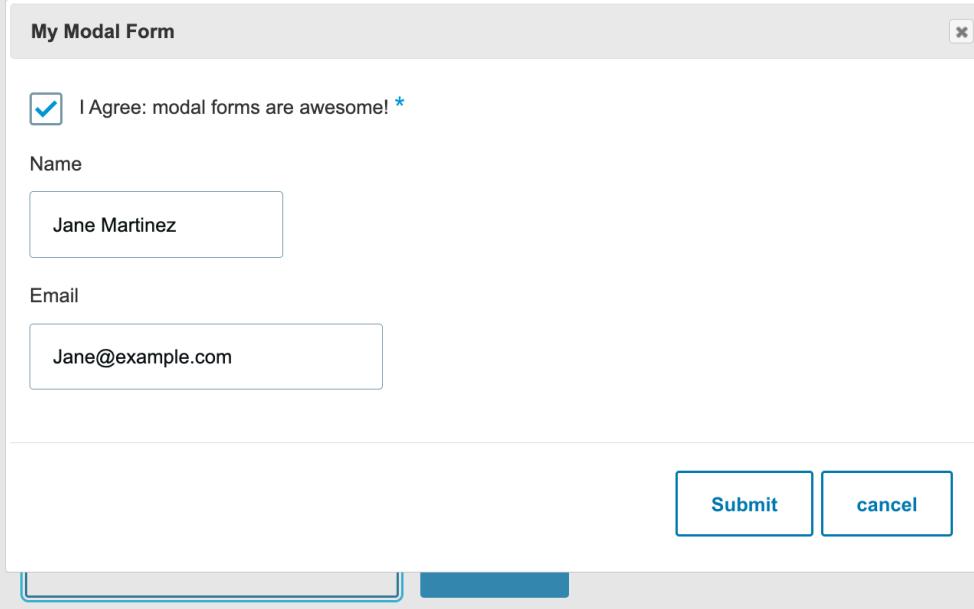
Name

Mary Vasquez

[Click to see the Modal Form](#)

[Submit](#)

And here is the modal popped up over it:



Here is the code for the first form in `web/modules/custom/modal_examples/src/Form/ExampleForm.php`. Notice the link at `$form['actions']['open_modal']` which is used to invoke the modal form. Note. It is essential that you specify the `use-ajax` class for this link.

```
<?php  
  
namespace Drupal\modal_examples\Form;  
  
use Drupal\Core\Form\FormBase;  
use Drupal\Core\Form\FormStateInterface;  
use Drupal\Core\Url;  
  
/**
```

```

 * ExampleForm class.
 */
class ExampleForm extends FormBase {

    /**
     * {@inheritDoc}
     */
    public function buildForm(array $form, FormStateInterface $form_state, $options = NULL) {

        $form['#prefix'] = '<div id="example_form">';
        $form['#suffix'] = '</div>';

        $form['info']['instructions'] = [
            '#type' => 'markup',
            '#markup' => $this->t('Please fill out the form below and click the button for more info.'),
        ];

        $form['info']['name'] = [
            '#type' => 'textfield',
            '#title' => $this->t('Name'),
            '#size' => 20,
            '#default_value' => 'Mary Vasquez',
            '#required' => FALSE,
        ];

        $form['actions']['open_modal'] = [
            '#type' => 'link',
            '#title' => $this->t('Click to see the Modal Form'),
            '#url' => Url::fromRoute('modal_examples.modal_form'),
            '#attributes' => [
                'class' => [
                    'use-ajax',
                    'button',
                ],
            ],
        ];
    }

    $form['actions']['submit'] = [
        '#type' => 'submit',
        '#value' => $this->t('Submit'),
        '#attributes' => [
        ],
    ];

    return $form;
}

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state) {
    // @TODO.
}

/**
 * {@inheritDoc}
 */
public function getFormId() {
    return 'modal_examples_example_form';
}

```

```
}
```

Here is the controller that is used to load the modal form at `web/modules/custom/modal_examples/src/Controller/ModalExamplesController2.php`

```
<?php

/**
 * @file
 *
 * Contains \Drupal\modal_examples\Controller\ModalExamplesController2.
 */
namespace Drupal\modal_examples\Controller;

use Symfony\Component\DependencyInjection\ContainerInterface;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\OpenModalDialogCommand;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Form\FormBuilder;

/**
 * ModalExamplesController2 class.
 */
class ModalExamplesController2 extends ControllerBase {

    /**
     * The form builder.
     *
     * @var \Drupal\Core\Form\FormBuilderInterface
     */
    protected $formBuilder;

    /**
     * The ModalFormExampleController constructor.
     *
     * @param \Drupal\Core\Form\FormBuilderInterface $formBuilder
     *   The form builder.
     */
    public function __construct(FormBuilderInterface $formBuilder) {
        $this->formBuilder = $formBuilder;
    }

    /**
     * {@inheritDoc}
     *
     * @param \Symfony\Component\DependencyInjection\ContainerInterface $container
     *   The Drupal service container.
     *
     * @return static
     */
    public static function create(ContainerInterface $container) {
        return new static(
            $container->get('form_builder')
        );
    }

    /**
     * Callback for opening the modal form.
     */
}
```

```

public function openModalForm() {
  $response = new AjaxResponse();

  // Get the modal form using the form builder.
  $modal_form = $this->formBuilder->getForm('Drupal\modal_examples\Form\ExampleModalForm');

  // Add an AJAX command to open a modal dialog with the form as the content.
  $response->addCommand(new OpenModalDialogCommand('My Modal Form', $modal_form, ['width' => '800']));

  return $response;
}

}

```

And finally, here is the "modal" form itself at `web/modules/custom/modal_examples/src/Form/ExampleModalForm.php`

```

<?php

namespace Drupal\modal_examples\Form;

use Drupal\Core\Ajax\CloseModalDialogCommand;
use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\OpenModalDialogCommand;
use Drupal\Core\Ajax\ReplaceCommand;

/**
 * ModalForm class.
 */
class ExampleModalForm extends FormBase {

  /**
   * {@inheritdoc}
   */
  public function getFormId() {
    return 'modal_form_example_modal_form';
  }

  /**
   * {@inheritdoc}
   */
  public function buildForm(array $form, FormStateInterface $form_state, $options = NULL) {
    $form['#prefix'] = '<div id="modal_example_form">';
    $form['#suffix'] = '</div>';

    // The status messages that will contain any form errors.
    $form['status_messages'] = [
      '#type' => 'status_messages',
      '#weight' => -10,
    ];

    // A required checkbox field.
    $form['our_checkbox'] = [
      '#type' => 'checkbox',
      '#title' => $this->t('I Agree: modal forms are awesome!'),
      '#default_value' => TRUE,
      '#required' => TRUE,
    ];
  }
}

```

```

$form['name'] = [
  '#type' => 'textfield',
  '#title' => $this->t('Name'),
  '#size' => 20,
  '#default_value' => 'Jane Martinez',
  '#required' => FALSE,
];

$form['email'] = [
  '#type' => 'email',
  '#title' => $this->t('Email'),
  '#size' => 30,
  '#default_value' => 'Jane@example.com',
  '#required' => FALSE,
];

$form['actions'] = array('#type' => 'actions');
$form['actions']['send'] = [
  '#type' => 'submit',
  '#value' => $this->t('Submit'),
  '#attributes' => [
    'class' => [
      'use-ajax',
    ],
    '#ajax' => [
      'callback' => [$this, 'submitModalFormAjax'],
      'event' => 'click',
    ],
  ],
];
$form['actions']['cancel'] = [
  '#type' => 'submit',
  '#value' => $this->t('cancel'),
  '#attributes' => [
    'class' => [
      'use-ajax',
    ],
    '#ajax' => [
      'callback' => [$this, 'closeModalForm'],
      'event' => 'click',
    ],
  ],
];
}

$form['#attached']['library'][] = 'core/drupal.dialog.ajax';

return $form;
}

/**
 * AJAX callback handler that displays any errors or a success message.
 */
public function submitModalFormAjax(array $form, FormStateInterface $form_state) {
  $response = new AjaxResponse();

  // If there are any form errors, re-display the form.
  if ($form_state->hasAnyErrors()) {
    $response->addCommand(new ReplaceCommand('#modal_example_form', $form));
  }
}

```

```

    }

    else {
        //Close the modal.
        $command = new CloseModalDialogCommand();
        $response->addCommand($command);

        // Pop up a new modal with a title success etc.
        $response->addCommand(new OpenModalDialogCommand("Success!", 'The modal form has been submitted.', ['width' => 800]));
    }

    return $response;
}

/**
 * @return \Drupal\Core\Ajax\AjaxResponse
 */
public function closeModalForm() {
    $command = new CloseModalDialogCommand();
    $response = new AjaxResponse();

    $response->addCommand($command);
    // $response->addCommand(new OpenModalDialogCommand("Cancel!", 'Why did you cancel me?', ['width' => 400]));

    return $response;
}

/**
 * {@inheritDoc}
 */
public function validateForm(array &$form, FormStateInterface $form_state) {
    // @TODO.
}

/**
 * {@inheritDoc}
 */
public function submitForm(array &$form, FormStateInterface $form_state) {
    // @TODO.
}

}

```

Slide-in dialog/Off-canvas dialog

Slide in dialogs are modals that appear to slide-in from the right side or top of the screen. Like any modal dialogs they can contain forms

Slide-in Dialog Example

This example shows 4 different types of links which fire off variations of slide-in dialogs. Check out the links below:

[Home](#)

Modal Examples (example2)



Route: *modal_examples.example2*

[Link to off canvas/slide-in custom modal dialog \(modal1\)](#)
[Link to off canvas/slide-in custom modal dialog \(modal1\) from top](#)
[Link to off canvas/slide-in form \(form2\)](#)
[Link to off canvas/slide-in login form \(login_form\)](#)

In `web/modules/custom/modal_examples/modal_examples.routing.yml` we specify the routes for our controller, `example2` as well as `modal1`, `form2` and `login_form`:

```

# Controller with buttons to open modals
modal_examples.example2:
path: '/modal-examples/example2'
defaults:
  _title: 'Modal Examples (example2)'
  _controller: '\Drupal\modal_examples\Controller\ModalExamplesController2::buildExample2'
requirements:
  _permission: 'access content'

# modal1
modal_examples.modal1:
path: '/modal-examples/modal1/{program_id}/{type}'
defaults:
  _title: 'Modal 1 with parameters'
  _controller: '\Drupal\modal_examples\Controller\ModalExamplesController::buildModal1'
requirements:
  _permission: 'access content'
options:
parameters:
  program_id:
  type:
no_cache: 'TRUE'

# Simple form for slide-in modals
modal_examples.form2:
path: '/modal-examples/form2'
defaults:
  _form: 'Drupal\modal_examples\Form\ExampleForm2'
  _title: 'Simple example form (form2)'
requirements:
  _permission: 'access content'

# Form with buttons to open modals
modal_examples.login_form:
path: '/modal-examples/login-form'
defaults:
  _title: 'Simple example form (form2)'
  _controller: 'Drupal\modal_examples\Controller\ModalExamplesController2::buildLoginForm'
requirements:
  _permission: 'access content'

```

Here is the controller for example2, `web/modules/custom/modal_examples/src/Controller/ModalExamplesController2.php`. You will find the functions `buildExample2()` which creates the 4 links and `buildLoginForm()` which uses Drupal's formbuilder to build the login form.

```

<?php

/**
 * @file
 *
 * Contains \Drupal\modal_examples\Controller\ModalExamplesController2.
 */
namespace Drupal\modal_examples\Controller;

use Drupal\Component\Serialization\Json;
use Drupal\Core\Ajax\OpenDialogCommand;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Drupal\Core\Url;
use Symfony\Component\DependencyInjection\ContainerInterface;

```

```

use Drupal\Core\Ajax\AjaxResponse;
use Drupal\Core\Ajax\OpenModalDialogCommand;
use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Form\FormBuilder;

/**
 * ModalExamplesController2 class.
 */
class ModalExamplesController2 extends ControllerBase {

  /**
   * The form builder.
   *
   * @var \Drupal\Core\Form\FormBuilderInterface
   */
  protected $formBuilder;

  /**
   * The entity type manager.
   *
   * @var \Drupal\Core\Entity\EntityTypeManagerInterface
   */
  protected $entityTypeManager;

  /**
   * The ModalFormExampleController constructor.
   *
   * @param \Drupal\Core\Form\FormBuilderInterface $formBuilder
   *   The form builder.
   * @param \Drupal\Core\Entity\EntityTypeManagerInterface $entity_type_manager
   *   The entity type manager.
   */
  public function __construct(FormBuilderInterface $formBuilder, EntityTypeManagerInterface $entity_type_manager) {
    $this->formBuilder = $formBuilder;
    $this->entityTypeManager = $entity_type_manager;
  }

  /**
   * {@inheritDoc}
   *
   * @param \Symfony\Component\DependencyInjection\ContainerInterface $container
   *   The Drupal service container.
   *
   * @return static
   */
  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('form_builder'),
      $container->get('entity_type.manager'),
    );
  }

  public function buildExample2() {

    $route_name = \Drupal::routeMatch()->getRouteName();
    $build['content'] = [
      '#type' => 'item',

```

```

"#markup' => $this->t('Route: %route', [%route' => $route_name]),
];

$build['link-to-modal1'] = [
  '#type' => 'link',
  '#prefix' => '<div class="pqrs1">',
  '#suffix' => '</div>',
  '#title' => t('Link to off canvas/slide-in custom modal dialog (modal1)'),
  '#url' => Url::fromRoute('modal_examples.modal1', [
    'program_id' => 123,
    'type' => 'all',
  ]),
  '#attributes' => [
    'id' => 'view-correlation-' . 12345,
    'class' => ['use-ajax'],
    'aria-label' => 'View useful information pertaining to item ' . '12345',
    '#prefix' => '<div class="abcdef">',
    '#suffix' => '</div>',
    'data-dialog-type' => 'dialog.off_canvas',
    'data-dialog-options' => Json::encode(
      [
        'width' => 'auto',
      ]
    ),
  ],
],
];

$build['link-to-modal1-top'] = [
  '#type' => 'link',
  '#prefix' => '<div class="pqrs1">',
  '#suffix' => '</div>',
  '#title' => t('Link to off canvas/slide-in custom modal dialog (modal1) from top'),
  '#url' => Url::fromRoute('modal_examples.modal1', [
    'program_id' => 123,
    'type' => 'all',
  ]),
  '#attributes' => [
    'id' => 'view-correlation-' . 12345,
    'class' => ['use-ajax'],
    'aria-label' => 'View useful information pertaining to item ' . '12345',
    '#prefix' => '<div class="abcdef">',
    '#suffix' => '</div>',
    'data-dialog-type' => 'dialog.off_canvas_top',
    'data-dialog-options' => Json::encode(
      [
        'width' => 'auto',
      ]
    ),
  ],
],
];

$build['link-to-example2-form'] = [
  '#type' => 'link',
  '#prefix' => '<div class="pqrs1">',
  '#suffix' => '</div>',
  '#title' => t('Link to off canvas/slide-in form (form2)'),
  '#url' => Url::fromRoute('modal_examples.form2', [
    'program_id' => 123,
    'type' => 'all',
  ]),
];

```

```

    ]),
    '#attributes' => [
      'id' => 'important-id' . 12345,
      'class' => ['use-ajax'],
      '#prefix' => '<div class="abcdef">',
      '#suffix' => '</div>',
      'data-dialog-type' => 'dialog.off_canvas',
      'data-dialog-options' => Json::encode(
        [
          {
            'width' => 'auto',
          }
        ],
      ),
    ],
  ];
}

$build['link-to-example-login-form'] = [
  '#type' => 'link',
  '#prefix' => '<div class="pqrsst">',
  '#suffix' => '</div>',
  '#title' => t('Link to off canvas/slide-in login form (login_form)'),
  '#url' => Url::fromRoute('modal_examples.login_form'),
  '#attributes' => [
    'id' => 'important-id' . 12345,
    'class' => ['use-ajax'],
    '#prefix' => '<div class="abcdef">',
    '#suffix' => '</div>',
    'data-dialog-type' => 'dialog.off_canvas',
    'data-dialog-options' => Json::encode(
      [
        {
          'width' => 'auto',
        }
      ],
    ),
  ],
];
}

$build['#attached']['library'][] = 'core/drupal.dialog.ajax';
$build['#attached']['library'][] = 'core/drupal.dialog.off_canvas';

return $build;
}

public function buildLoginForm() {
  // $form = \Drupal::formBuilder()->getForm(\Drupal\user\Form\UserLoginForm::class);
  $form = $this->formBuilder()->getForm(\Drupal\user\Form\UserLoginForm::class);
  return $form;
}
}

```

BuildModal1() is in ModalExamplesController.php (above):

```

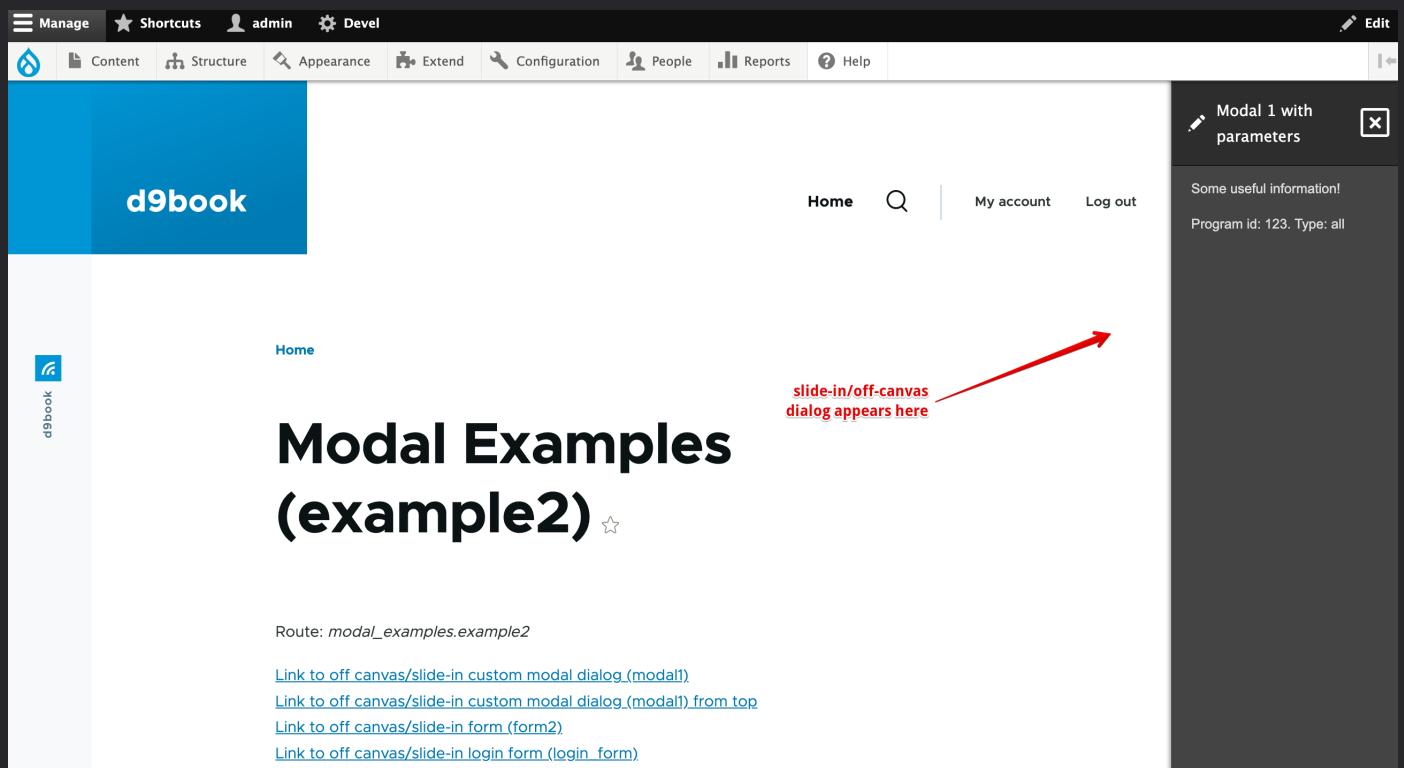
public function buildModal1(int $program_id, string $type) {

    $build['content'] = [
        '#type' => 'item',
        '#markup' => $this->t('Some useful information!'),
    ];
    $build['other_content'] = [
        '#type' => 'item',
        '#markup' => $this->t('Program id: @program_id. Type: @type', [
            '@program_id' => $program_id,
            '@type' => $type,
        ]),
    ];
}

return $build;
}

```

This is what to expect when you click the first link:



And the second which shows the dialog sliding in from the top:

A screenshot of a Drupal website titled "Modal Examples (example2)". At the top, there is a dark modal dialog box with the title "Modal 1 with parameters". Inside the modal, there is some text: "Some useful information!" and "Program id: 123. Type: all". Below the modal, the main content area shows the page title "Modal Examples (example2)" and a "Route: modal_examples.example2". A red arrow points upwards from the bottom of the modal to the top of the screen, indicating the direction it slid in from.

Here is our custom form sliding in:

A screenshot of a Drupal website titled "(example2)". On the right side of the page, a modal dialog box titled "Simple example form (form2)" is displayed. The dialog contains a form with a single input field labeled "Name" containing "Mary Vasquez" and a "Submit" button. A red arrow points from the left edge of the modal towards the center of the page, indicating the direction it slid in from the side.

And finally the login form sliding in:

The screenshot shows a Drupal administrative interface. At the top, there's a toolbar with various links like 'MISC', 'Clipboard', 'Clients', etc. Below the toolbar, the main navigation bar includes 'Manage', 'Shortcuts', 'admin', 'Devel', and other options like 'Content', 'Structure', 'Appearance', 'Extend', 'Configuration', 'People', 'Reports', and 'Help'. A blue sidebar on the left has a menu icon. The main content area has a title '(example2)' and some text links. On the right, a modal dialog titled 'Simple example form (form2)' is open, containing fields for 'Username' (with value 'admin') and 'Password' (with value '*****'). A red arrow points from the text 'login form' in the main content area to the 'Log In' button in the modal.

Block with a link to popup a custom modal dialog

This is an example from <http://befused.com/drupal/modal-module>

In the module `custom_modal`, there is a `custom_modal.routing.yml` which specifies the route and controller.

```
custom_modal.modal:
  path: 'modal-example/modal'
  defaults:
    _title: 'Modal'
    _controller: '\Drupal\custom_modal\Controller\CustomModalController::modal'
  requirements:
    _permission: 'access content'
```

Here we define a block in `ModalBlock.php` that will present a button, which when clicked, will bring up a modal dialog. Note in the `build()` method that we get the route using `Url::fromRoute('custom_modal.modal')`.

```

/**
 * Provides a 'Modal' Block
 *
 * @Block(
 *   id = "modal_block",
 *   admin_label = @Translation("Modal block"),
 * )
 */
class ModalBlock extends BlockBase {

  /**
   * {@inheritDoc}
   */
  public function build() {
    $link_url = Url::fromRoute('custom_modal.modal');
    $link_url->setOptions([
      'attributes' => [
        'class' => ['use-ajax', 'button', 'button-small'],
        'data-dialog-type' => 'modal',
        'data-dialog-options' => Json::encode(['width' => 400]),
      ]
    ]);
    return array(
      '#type' => 'markup',
      '#markup' => Link::fromTextAndUrl(t('Open modal'), $link_url)->toString(),
      '#attached' => ['library' => ['core/drupal.dialog.ajax']]
    );
  }
}

```

Then in the controller, CustomModalController.php the modal() function builds the dialog using a new AjaxResponse() object.

```

class CustomModalController extends ControllerBase {

  public function modal() {
    $options = [
      'dialogClass' => 'popup-dialog-class',
      'width' => '50%',
    ];
    $response = new AjaxResponse();
    $response->addCommand(new OpenModalDialogCommand(t('Modal title'), t('The modal text'), $options));

    return $response;
  }
}

```

To see this work, place the “modal block” in a region via the u/i, then view a page. When you click the button, a modal dialog will pop up showing “the modal text.”

No-code modal dialogs

Modal dialogs are incredibly useful on websites as they allow the user to do something without having to leave the web page they are on. Drupal has a Dialog API in core, which greatly reduces the amount of code you need to write to create a modal dialog. Dialogs in Drupal 8 leverage jQuery UI. This is from an article at <http://befused.com/drupal/modal>.

This capability puts some awesome power in the hands of site builders.

In a custom block with full HTML enabled use the following HTML.

```
<p><a class="use-ajax" data-dialog-type="modal" href="/search/node">Search</a></p>
```

Or to make this dialog a little wider. E.g. 800 pixels.

```
<p><a class="use-ajax" data-dialog-options="{"width":800}" data-dialog-type="modal" href="/search/node">Search</a></p>
```

And to show a node instead of the search dialog (any valid drupal path will do here)

```
<p><a class="use-ajax" data-dialog-options="{"width":800}" data-dialog-type="modal" href="/node/2">See node 2</a></p>
```

Resources

- [Excellent tutorial on using modal forms in Drupal from March 2014](#)
 - [Phil Norton's great article on Creating AJAX dialogs from October 2022 at](#)
 - [Article about creating a modal dialog in a custom module from Jan 2018](#)
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) 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.

Nodes and Fields

TABLE OF CONTENTS

- [Load a node and get a formatted text field](#)
- [Load a numeric field value](#)
- [Set field values](#)
- [Get current page title](#)
- [Test if variable is a node](#)
- [Get the current nid, node type and title](#)
- [Retrieve current node id \(nid\)](#)
- [Retrieve node info from current path](#)
- [Load the current node and get it's node id \(nid\), field, type](#)
- [Load a node by nid and get its title, type and a field](#)
- [Load the current node and get the nid, field, type](#)
- [Load the user id \(uid\) for a node](#)
- [Test if a field is empty](#)
- [Load a node and update a field](#)
- [Load values from a date range field](#)
- [Load multivalue field](#)
 - [Iterate through results](#)
 - [Read a specific instance](#)
- [Update a multivalue field](#)
 - [Function to read and write multivalue fields](#)
 - [Save multivalue field, entity reference field](#)
 - [Update a multivalue entity reference fields](#)
 - [Generic Multivalue field writer](#)
- [Does this field exist in my entity?](#)
- [Get URL for an image or file in a media reference field](#)
- [Retrieve info about a file field](#)
- [Retrieve a link field](#)
- [Does this field exist in my entity?](#)
- [Create a node and write it to the database](#)
- [Create a node with an image](#)
- [Write a node with an attached file](#)
- [Write a date or datetime to a node](#)
- [Or just a date \(no time\)](#)
- [Set field values](#)
- [Set an entity reference field](#)
- [Set multivalue fields \(regular and entity reference\)](#)
- [Clear a text field](#)
- [Set or clear a body field](#)
- [Load a node and retrieve an entity reference node and nid \(target_id\)](#)
 - [Load a multivalue reference field.](#)
- [Entity reference nodes and their fields](#)
- [Load the taxonomy terms from a term reference field](#)
- [Load a node and find the terms referenced in a paragraph in a term reference field](#)
- [Retrieve a URL field](#)
 - [External links](#)

- [Internal links](#)
- [Load a node and retrieve a paragraph field](#)
- [How to get Node URL alias or Taxonomy Alias by Node id or Term ID](#)
- [How to set a URL Alias](#)
- [Get a node's menu item and more](#)
- [Find a node using it's uid](#)
- [Retrieve Node ID\(NID\) or Taxonomy term ID from a Drupal alias or path](#)
- [Retrieve all nodes with a matching taxonomy term](#)
- [How to uncache a particular page or node](#)
- [Get boolean Field](#)
- [Date Field](#)
- [Date Fields](#)
- [Date Range Field doesn't display correct timezone](#)
- [Date Range](#)
- [Date Range fields: Load start and end values](#)
- [Date Fields: Load or save them](#)
- [Comparing DrupalDateTime values](#)
- [Date with embedded timezone](#)
- [Has something expired?](#)
- [Load or save Drupal Date fields](#)
- [Retrieve node creation date and format it](#)
- [Retrieve node creation or changed date and format it](#)
- [Date Field and Date with no time \(remove time\)](#)
- [Smart date \(smart_date\) load and format](#)
- [Smart date \(smart_date\) all-day](#)
- [Smart date \(smart_date\) range of values](#)
- [hook_node_presave or hook_entity_type_presave](#)
- [Disable caching for a content type](#)
- [Writing some JSON data into a long text field](#)
- [Create a node with an image](#)
- [Paragraphs](#)
- [Load a node and find the terms referenced in a paragraph in a term reference field](#)
- [Custom Field Formatter](#)
- [Puzzles](#)
 - [What can I do with a call to first\(\) on an entity reference field?](#)
- [Great Cheat sheets](#)

views 178

Load a node and get a formatted text field

```
$nodeStorage = $this->entityTypeManager->getStorage('node');
,node = $nodeStorage->load(10706);
$body = $node->get('body')->value;
$body = $node->body->value;

//after text filters have done their magic!
$body = $node->body->processed;
```

Load a numeric field value

When you load a numeric field, Drupal returns a number i.e. 0 even if that field was never initialized with a value.

```
$accepted_votes = $feedback_error_node->get('field_accepted_votes')->value;  
// Returns 0 if no value was entered into the field.
```

Set field values

```
$node->set('field_tks_subchapter', $subchapterFullStatement);  
$node->set('field_tks_standard_type', "TEKS");  
$node->save();
```

Get current page title

Use this in a controller, to return the current page title.

```
$request = \Drupal::request();  
if ($route = $request->attributes->get(\Symfony\Component\Routing\RouteObjectInterface::ROUTE_OBJECT)) {  
    $title = \Drupal::service('title_resolver')->getTitle($request, $route);}
```

Test if variable is a node

```
// Is this variable a node?  
if ($ref instanceof EntityInterface && $ref->getEntityTypeId === 'node') {  
    // yep  
}
```

Get the current nid, node type and title

Here are two ways to retrieve the current node – via the request or using the route

```
$node = \Drupal::request()->attributes->get('node');  
$nid = $node->id();
```

OR

```
$node = \Drupal::routeMatch()->getParameter('node');  
if ($node instanceof \Drupal\node\NodeInterface) {  
    // You can get nid and anything else you need from the node object.  
    $nid = $node->id();  
    $nodeType = $node->bundle();  
    $nodeTitle = $node->getTitle();  
}
```

If you need to use the node object in `hook_preprocess_page()` on the preview page, you need to use the `node_preview` parameter, instead of the `node` parameter:

```
function mymodule_preprocess_page(&$vars) {  
  
    $route_name = \Drupal::routeMatch()->getRouteName();  
  
    if ($route_name == 'entity.node.canonical') {  
        $node = \Drupal::routeMatch()->getParameter('node');  
    }  
    elseif ($route_name == 'entity.node.preview') {  
        $node = \Drupal::routeMatch()->getParameter('node_preview');  
    }  
}
```

And from <https://drupal.stackexchange.com/questions/145823/how-do-i-get-the-current-node-id> when you are using or creating a custom block then you have to follow this code to get current node id. Not sure if it is correct

```

use Drupal\Core\Cache\Cache;

$node = \Drupal::routeMatch()->getParameter('node');
if ($node instanceof \Drupal\node\NodeInterface) {
  $nid = $node->id();
}

// for cache
public function getCacheTags() {
  //With this when your node changes your block will rebuild
  if ($node = \Drupal::routeMatch()->getParameter('node')) {
    //if there is node add its cachetag
    return Cache::mergeTags(parent::getCacheTags(), ['node:' . $node->id()]);
  }
  else {
    //Return default tags instead.
    return parent::getCacheTags();
  }
}

public function getCacheContexts() {
  //if you depend on \Drupal::routeMatch()
  //you must set context of this block with 'route' context tag.
  //Every new route this block will rebuild
  return Cache::mergeContexts(parent::getCacheContexts(), ['route']);
}

```

Retrieve current node id (nid)

If the site is currently displaying node/123, this will return a path node/123.

```
$current_path = \Drupal::service('path.current')->getPath();
```

Also

```

$node = \Drupal::routeMatch()->getParameter('node');
if ($node instanceof \Drupal\node\NodeInterface) {
  // You can get nid and anything else you need from the node object.
  $nid = $node->id();
  $nodeTitle = $node->getTitle();
  $nodeType = $node->bundle();
  if ($nodeType != 'unit') {
    return;
  }
}

```

Retrieve node info from current path

You don't need to load the node

```

$node = \Drupal::routeMatch()->getParameter('node');
if ($node instanceof \Drupal\node\NodeInterface) {
  // You can get nid and anything else you need from the node object.
  $nid = $node->id();
  if ($node->getType() == 'unit') {
    $contract_id = $node->field_contract_id->value;
    $facebook = $node->field_facebook->value;
  }
}

```

Load the current node and get it's node id (nid), field, type

To grab some information from the currently displayed node, use the \Drupal object::routeMatch().

```
// Grab current node.  
$node = \Drupal::routeMatch()->getParameter('node');  
if ($node instanceof \Drupal\node\NodeInterface) {  
    // You can get nid and anything else you need from the node object.  
    $nid = $node->id();  
    if ($node->getType() == 'unit') {  
        $v = $node->get('field_footer_social_media_headin')->value;  
        if (!empty($v)) {  
            ...  
        }  
        $v = $node->get('field_facebook')->value;  
        if (!empty($v)) {  
            ...  
        }  
    }  
}
```

Note you can also `$node->get('field_facebook')->getValue()` which returns an array of values. This is a good way to retrieve unlimited value fields i.e. fields that have more than 1 value.

Load a node by nid and get its title, type and a field

From: <https://www.drupal.org/docs/8/api/entity-api/working-with-the-entity-api>

```
use Drupal\node\Entity\Node;  
// Using the node id = 1.  
$node = Node::load(1);
```

Or

```
$node = \Drupal::entityTypeManager()->getStorage('node')->load(1);  
$headline = $node->getTitle();  
  
// What type of node is that?  
$type = $node->getType();  
  
// Get the url field.  
$url = $node->get('field_url')->value;
```

Or

```
$node_storage = \Drupal::entityTypeManager()->getStorage('node');  
$node = $node_storage->load(1);  
$url = $node->get('field_url')->value;
```

Or

```
$body = $node->field_srp_pc_change_type->value;
```

And multivalue fields are similar. They are loaded when you use `get & getValue` or the magic getter & `getValues`. You can then loop through the returned values:

```
$topics = $node->get('field_news_topics_for_listing')->getValues(); //get.
```

or

```
$topics = $node->field_news_topics_for_listing->getValue(); //magic getter.

foreach ($topics as $topic) {
    // Do something.
}
```

Load the current node and get the nid, field, type

```
// For content type unit grab their facebook/instagram etc.
$node = \Drupal::routeMatch()->getParameter('node');

if ($node instanceof \Drupal\node\NodeInterface) {

    // You can get nid and anything else you need from the node object.

    $nid = $node->id();

    if ($node->getType() == 'unit') {
        $sm_heading = $node->get('field_footer_social_media_headin')->value;

        if (!empty($sm_heading)) {
            ...
        }
    }

    $fb = $node->get('field_facebook')->value;

    if (!empty($fb)) {
        ...
    }
}
```

Note you can also

```
$node->get('field_facebook')->getValue();
```

Which returns an array of values.

Load the user id (uid) for a node

```
$my_node->getOwnerId();
```

You can call `$node->uid` but that returns an `EntityReferenceFieldItemList` with all sorts of juicy information. The user id is in there but more challenging to extract.

Test if a field is empty

```
$entity->get('field_name')->isEmpty()
```

For an entity field use:

```
$sf_contract_node = Node::load($sf_contract_nid);

if ($sf_contract_node) {
    if (!$sf_contract_node->get('field_vendor_url')->isEmpty()) {
        $url = $sf_contract_node->field_vendor_url->first()->getUrl();
        $uri = $url->getUri();
        $variables['vendor_url'] = $uri;
    }
}
```

And more concisely, using a new feature of PHP 8 we can use the following:

```
$url = $sf_contract_node?->field_vendor_url?->first()?->getUrl();

if (!is_null($url)) {
    $uri = $url->getUri();
}
```

Load a node and update a field

```
$node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
$url = $node->set('field_url', 'the-blahblah');
,node->save();
```

Load values from a date range field

Start date and then end date

```
$node->get('field_cn_start_end_dates')->value
,$node->get('field_cn_start_end_dates')->end_value
```

Load multivalue field

Multivalue fields can be loaded with `get('fieldname')` or using a magic field getter like `$node->field_my_field`. Adding `->getValue()` to the end of either of these calls returns a simple array.

For example, a multivalue text field, this will return an array of values like this:

```
$data = $node->get('field_condiment')->getValue();
// returns:
$data[0]['value'] = 'ketchup'
$data[1]['value'] = 'mayo'
$data[2]['value'] = 'reference'
```

However a multivalue entity reference field (including taxonomy) will return values with the `target_id` array key. e.g.

```
$data = $node->get('field_event_ref')->getValue();
// returns:
$data[0]['target_id'] = 1
$data[1]['target_id'] = 2
$data[2]['target_id'] = 14
```

Iterate through results

Using `$node->field_condiment` OR `$node->get('field_condiment')` returns a `Drupal\Core\Field\FieldItemList` which is iterable. You can loop through the results and retrieve the values like this:

```
$items = $node->field_condiment;
foreach ($items as $item) {
  $result = $item->value;
  // For entity reference fields use
  // $result_nid = $item->target_id;
}
```

For entity reference fields, use `target_id` rather than `->value`.

And you can check if there is a particular item in the array like this:

```
// get FieldItemList.
$condiments = $node->get('field_condiment');
$vote_number = 1;
if (isset($condiments[$vote_number])) {
  $result = $condiments[$vote_number]->value;
}
// $result is ketchup.

// Or the PHP 8 way:

$result = $node->field_condiment?->get($vote_number)?->value;
// OR
$result = $node->field_condiment[$vote_number]?->value;
```

Read a specific instance

You can directly reference an item by specifying an array offset. The index key (0 or 1 below) can also be referred to as the delta.

```
$status = $node->get('field_voting_status')[0]->value;
$status = $node->get('field_voting_status')[1]->value;
```

Note. If the value for delta 1 is empty, Drupal will throw a warning message *Warning*: Attempt to read property "value" on null in ...

So rather than reading the [1]->value directly, you should check if there is a value using `isset()` and then, you can read the `->value`. Note. When you do the `isset()` test, you don't add the `->value` at the end e.g.

```
if(!is_null($node->get('field_voting_status')[$vote_number])) {
  $voting_status = $correlation_node->field_voting_status[$vote_number]->value;
}
```

Update a multivalue field

This can be a little tricky especially if you want to preserve the existing values in the field.

Here I used the `getValue()` to load all the values. This returns an array of values like:

```
$data = $node->get('field_event_ref')->getValue();

$result0 = $data[0]['value'];
$result1 = $data[1]['value'];
```

So to update one of them, I want to do something like this: `$data[1]['value'] = 'flour';`.

e.g. I load up the current values, fill in the 6th item and save them. This preserves the existing values in positions [0]through [4].

```
$values = $node->get('field_voting_status')->getValue();
$values[5]['value'] = 'incomplete';
$node->set('field_srp_voting_status', $values);
$node->save();
```

Function to read and write multivalue fields

Here is a function which reads and writes multivalue fields safely. You pass it the `$node->field_name`, the index (`vote_number`) etc. and then it builds and returns an array formatted for updating the field data. It can also update the array if you pass in a value.

It could probably be genericized further.

```

public static function getMultivalueFieldArrayOfValues($values_field_data, $vote_number = 0, $new_value = NULL, $field_type = 'value', $default_value = NULL) {
    $values_array = [];
    foreach($values_field_data as $data) {
        $values_array[] = [
            $field_type => $data->value,
        ];
    }
    if(!$is_null($new_value)) {
        //CHECK FOR PREVIOUS VALUES. IF THEY DO NOT EXIST SET TO NEW VALUE
        for($i=0; $i<=$vote_number; $i++) {
            if(!isset($values_array[$i])) {
                if(!$is_null($default_value)) {
                    $values_array[$i] = $default_value;
                } else {
                    $values_array[$i] = $new_value;
                }
            }
        }
        if(isset($values_array[$vote_number])) {
            $values_array[$vote_number] = [$field_type => $new_value];
        } else {
            $values_array[] = [$field_type => $new_value];
        }
    }
    return $values_array;
}

```

Example of using the above function for reading data and then writing the data without using it:

```

// Retrieve current field values for correlation narrative/activity status fields

$activity_status = 'accepted';
$activity_statuses = $node->field_activity_status;
$activity_status_values = self::getMultivalueFieldArrayOfValues($activity_statuses);
$activity_status_values[0]['value'] = $activity_status;

$narrative_status = 'accepted';
$narrative_statuses = $node->field_narrative_status;
$narrative_status_values = self::getMultivalueFieldArrayOfValues($narrative_statuses);
$narrative_status_values[0]['value'] = $narrative_status;

$node->set('field_narrative_status', $narrative_status_values);
$node->set('field_activity_status', $activity_status_values);
$node->save();

```

Save multivalue field, entity reference field

When you really care which delta/index/offset, you can specify that offset. It's a bit confusing how exactly it works. For text or numeric fields, it works like you'd expect. You can just specify the offset. For entity reference fields, you have to do some fiddling. Don't use `$node->set()` as this overwrites everything in the field, rather use the magic field setter variable and specify the offset.

Here `$vote_number` represents the index so if `$vote_number = 0`, this write the first item in the multivalue field. If `$vote_number = 1`, then write the second item, and so on.

```

$citation_node->field_srп_voting_status[$vote_number] = 'incomplete';

```

Be cautious, you might think this would work but it *doesn't*

```
$program_node->set('field_srp_team_ref', [$vote_number => 1234];
```

If you are going to write index 2 and there is a possibility that there isn't an index 0 and 1, you need something like this (in `protected function correlationSanityCheckFix(Node $correlation_node)`):

```
$narrative_status = '';

if (!is_null($correlation_node->get('field_narrative_status')[$vote_number])) {
  $narrative_status = $correlation_node->get('field_narrative_status')[$vote_number]->value;
}

if (empty($narrative_status)) {
  // Grab all the statuses and add mine.
  $statuses = $correlation_node->get('field_narrative_status');
  $new_statuses = [];
  foreach ($statuses as $status) {
    $new_statuses[] = $status->value;
  }
  $new_statuses[$vote_number] = 'incomplete';
  $correlation_node->set('field_narrative_status', $new_statuses);
  $save_correlation = TRUE;
}
```

Update a multivalue entity reference fields

When writing multivalue entity reference fields, you have to load up the previous values, build an array of `target_ids` (node ids) and then write them all in one pass with a `$node->set()` method.

```
$teams = $new_program_node->get('field_srp_team_ref');
$new_teams = [];
foreach ($teams as $team) {
  $new_teams[] = $team->target_id;
}
$new_teams[$new_program_vote_number] = $team_id;
$new_program_node->set('field_srp_team_ref', $new_teams);
```

Generic Multivalue field writer

Here is a generic function that knows how to write values in a “sane” way.

```
/** 
 * Smart multi value field setter.
 *
 * Example calls:
 *
 * Set the index 2 to incomplete, keep old values:
 * smartMultiValueFieldSetter($node, 'field_srp_voting_status', 'incomplete', 2);
 *
 * Set the index 1 to incomplete, overwrite the old values to 'placeholder'
 * smartMultiValueFieldSetter($node, 'field_srp_voting_status', 'incomplete', 1, 'placeholder', TRUE);
 *
 *
 * @param \Drupal\node\Entity\Node $node
 *   Node.
 * @param string $field_name
 *   Field name.
 * @param string $value
 *   Value to be put in $node->field[$index]->value.
 * @param int $index
```

```

* The delta i.e. $node->field[$index]
* @param string $default_value
* The default values that will be written into the previous indexes.
* @param bool $overwrite_old_values
* TRUE to ignore previous index values and overwrite them with $default_value.
*/
public static function smartMultiValueFieldSetter(Node $node, string $field_name, string $value, int $index, string $default_value="", bool $overwrite_old_values=False) {
$old_values = $node->get($field_name)->getValue();

// Grab old values and put them into $new_values array.

$field_type = $node->get($field_name)->getFieldDefinition()->getType();
if ($field_type == 'entity_reference') {
foreach ($old_values as $key=>$old_value) {
$new_values[$key] = $old_values[$key];
}
}
else {
$new_values = [];
foreach ($old_values as $old_value) {
$new_values[]["value"] = $old_value["value"];
}
}

// Ignore what was in the old values and put my new default value in.

if ($overwrite_old_values) {
for ($i = 0; $i < $index; $i++) {
$new_values[$i] = $default_value;
}
}

// Pad missing items.

for ($i = 0; $i < $index; $i++) {
if (!isset($new_values[$i])) {
if ($field_type == 'entity_reference') {
$new_values[$i] = $default_value;
}
else {
$new_values[$i]['value'] = $default_value;
}
}
}

if ($field_type == 'entity_reference') {
$new_values[$index]['target_id'] = $value;
}
else {
$new_values[$index]["value"] = $value;
}

// Trim off extras from testing.

// TODO: this isn't trimming correctly for entity ref fields.

if (count($new_values) > ($index+1)) {
$chunk = array_chunk($new_values, $index+1);
$new_values = $chunk[0];
}

$node->set($field_name, $new_values);
}

```

Here is an example of using the above function to write values to the field_condiment which is a multivalue text field.

```
$node = Node::load(35);

// Write to index 0, 1, 2.
self::smartMultiValueFieldSetter($node, 'field_condiment', 'ketchup', 0);
self::smartMultiValueFieldSetter($node, 'field_condiment', 'mayo', 1);
self::smartMultiValueFieldSetter($node, 'field_condiment', 'mustard', 2);
$node->save();

// Write into index position 2 and pad the previous values with "dummy".
self::smartMultiValueFieldSetter($node, 'field_condiment', 'mustard', 2, 'dummy', TRUE);
$node->save();

// Write into index position 1 and pad position 0 with "dummy", and remove the contents of index 2.
self::smartMultiValueFieldSetter($node, 'field_condiment', 'ketchup', 1, 'dummy', TRUE);
$node->save();
```

Here is a complete function from the controller GeneralController.php. There are a wide variety of calls to smartMultiValueFieldSetter() showing it's use with multivalue text and entity-reference fields (including a taxonomy field):

```
public function multiTest() {

  $str = '<h2>Results</h2>';

  $node = Node::load(35);

  // Write to index 0, 1, 2.
  self::smartMultiValueFieldSetter($node, 'field_condiment', 'ketchup', 0);
  self::smartMultiValueFieldSetter($node, 'field_condiment', 'mayo', 1);
  self::smartMultiValueFieldSetter($node, 'field_condiment', 'mustard', 2);
  // $node->save();

  $field_name = 'field_condiment';
  $field_type = $node->get($field_name)->getFieldDefinition()->getType();

  $contents = $node->get($field_name)->getValue();
  $str .= "<br/><strong>Field:</strong> " . $field_name;
  $str .= ", type: " . $field_type;
  $str .= "<br/><strong>Values: </strong>";
  foreach ($contents as $item) {
    $str .= $item['value'] . ',';
  }

  self::smartMultiValueFieldSetter($node, 'field_condiment', 'mustard', 2, 'dummy', TRUE);
  $contents = $node->get($field_name)->getValue();
  $str .= "<br/><strong>Values: </strong>";
  foreach ($contents as $item) {
    $str .= $item['value'] . ',';
  }

  self::smartMultiValueFieldSetter($node, 'field_condiment', 'ketchup', 1, 'dummy', TRUE);
  $contents = $node->get($field_name)->getValue();
  $str .= "<br/><strong>Values: </strong>";
  foreach ($contents as $item) {
    $str .= $item['value'] . ',';
  }
}
```

```

$field_name = 'field_event';
$field_type = $node->get($field_name)->getFieldDefinition()->getType();
$contents = $node->get($field_name)->getValue();
//kint($contents);
$str .= "<br><strong>Field:</strong> " . $field_name;
$str .= ", type: " . $field_type;
$str .= "<br><strong>Values: </strong>";
foreach ($contents as $item) {
  $str .= $item['target_id'] . ',';
}

// 17, 18
//14, 18, 19
self::smartMultiValueFieldSetter($node, $field_name, 14, 0);
self::smartMultiValueFieldSetter($node, $field_name, 18, 1);
self::smartMultiValueFieldSetter($node, $field_name, 19, 2);
$contents = $node->get($field_name)->getValue();
$str .= "<br><strong>Values: </strong>";
foreach ($contents as $item) {
  $str .= $item['target_id'] . ',';
}

$field_name = 'field_category';
$field_type = $node->get($field_name)->getFieldDefinition()->getType();
$contents = $node->get($field_name)->getValue();
//kint($contents);
$str .= "<br><strong>Field:</strong> " . $field_name;
$str .= ", type: " . $field_type;
$str .= "<br><strong>Values: </strong>";
foreach ($contents as $item) {
  $str .= $item['target_id'] . ',';
}

// 3, 2, 1
// 1, 2, 3.
self::smartMultiValueFieldSetter($node, $field_name, 1, 0);
self::smartMultiValueFieldSetter($node, $field_name, 2, 1);
self::smartMultiValueFieldSetter($node, $field_name, 3, 2);
$contents = $node->get($field_name)->getValue();
//kint($contents);
$str .= "<br><strong>Values: </strong>";
foreach ($contents as $item) {
  $str .= $item['target_id'] . ',';
}

// Multivalue Text Field.
$field_name = 'field_condiment';
// Returns FieldItemList.
$data = $node->get($field_name);
$data = $node->field_condiment;
// Returns simple array.
$data = $node->get($field_name)->getValue();
$data = $node->field_condiment->getValue();

// Multivalue entity reference field.
$field_name = 'field_event';
$data = $node->get($field_name)->getValue();
$data = $node->field_event;

```

```

// Multivalue taxonomy entity reference field.
$field_name = 'field_category';
$data = $node->get($field_name)->getValue();
// Yes, you can use a variable for a magic field getter!
$data = $node->$field_name;

// Loop thru results.
$items = $node->field_condiment;
foreach ($items as $item) {
  $x = $item->value;
}

// get array of results.
$condiments = $node->get('field_condiment');
$vote_number = 1;
if (isset($condiments[$vote_number])) {
  $result = $condiments[$vote_number]->value;
}
$result = $node->get('field_condiment')[0]->value;
$result = $node->get('field_condiment')[5]->value;
if (isset($node->get('field_condiment')[2]->value)) {
  $result = $node->get('field_condiment')[2]->value;
}

$render_array['content'] = [
  '#type' => 'item',
  '#markup' => $str,
];
return $render_array;
}

```

Does this field exist in my entity?

To check if a field exists in a node.

```
$entity->hasField('abc');
```

e.g. Here we load a field `field_library_media` from a node, grab it's target id (which we happen to know is a media entity.). We load the media entity and check if there is a field called `field_media_document`. This rather convoluted example is used to get the file size of the file in the media field.

```

use Drupal\media\Entity\Media;
use Drupal\file\Entity\File;

$media_id = $node->field_library_media->target_id;
if ($media_id) {
  $media_item = Media::load($media_id);
  // Get the file.
  if ($media_item->hasField('field_media_document')) {
    $file_id = $media_item->field_media_document->getValue()[0]['target_id'];
  }
  if (isset($file_id)) {
    $file = File::load($file_id);
    if ($file) {
      // Get file size.
      $file_size = format_size($file->getSize());
      // Set file size variable.
      $variables['file_size'] = $file_size;
    }
  }
}

```

Get URL for an image or file in a media reference field

```

$node = Node::load($nid);
if ($node) {
  $id = $node->field_banner_image->target_id;
  $media = Media::load($id);
  //Not always correct
  // $fid = $media->field_media_image->target_id;
  //Better way:
  $fid = $media->getSource()->getSourceFieldValue($media);
  $file = File::load($fid);
  if ($file) {
    $uri = $file->getFileUri(); //uri e.g. public://2020-12/atom.jpg

    $media_url = file_create_url($uri); //Full URL of uploaded image.
    //E.g. https://dir.ddev.site/sites/default/files/2020-12/atom.jpg
  }
  \Drupal::messenger()->addMessage("field_banner image url is $media_url");
  $variables['program_area_banner_image_url'] = $media_url;
}

```

You can get relative or absolute paths using one of these

```

$file = File::load($fid);
if ($file) {
  $uri = $file->getFileUri(); //returns drupal filename e.g. public://2020-12/atom.jpg

  $file_url = file_create_url($uri); //returns absolute file url e.g. https://...file.jpg

  //also
  $absolute_file_url = $file->createFileUrl(FALSE); //returns absolute file path (url)

  $relative_file_url = $file->createFileUrl(TRUE); //returns relative file path (url)
}

```

Retrieve info about a file field

Here we have a file field called `field_materials_file` that we loaded from a paragraph. Looking in `core/modules/file/src/Entity/File.php` we can see a series

of useful functions like `getFileName()`, `getFileUri()`, `getSize()`, `getCreatedTime()`. To use these, we have to append `->entity` after the field as shown below

```
$file = $para->field_materials_file;  
$filename = $file->entity->getFileName();  
$uri = $file->entity->getFileUri();
```

Since `File` is an entity, we can also look in `EntityBase.php` to find more useful functions like `id()`, `label()`, `bundle()`.

Retrieve a link field

Here we have a link field: `field_link` which we load and get a valid uri from it using:

```
$link = $para->field_link;  
$link_uri = $para->field_link->uri;
```

Or

```
$current_url = $correction->get('field_link')->uri;
```

Or

In a `.module` file, `first()` returns a `Drupal\Link\Plugin\FieldType\LinkItem`

```
if ($sf_contract) {  
  $vendor_url = $sf_contract->field_vendor_url->first();  
  if ($vendor_url) {  
    // this works too: $vendor_url = $vendor_url->uri;  
    $vendor_url = $vendor_url->getUrl(); // returns a Drupal\Core\Url.  
    $vendor_url = $vendor_url->toString();  
    _add_single_metatag("vendor_url", $vendor_url, $variables);  
  }  
}
```

Leaving off the `->first()` (like this) returns a `Drupal\Core\Field\FieldItemList` which is a list of fields so you then would have to pull out the first field and extract the uri out of that.

```
$vendor_url = $sf_contract->field_vendor_url;
```

Does this field exist in my entity?

```
$entity->hasField('abc');
```

e.g. Here we load a field `field_library_media` from a node, grab its target id (which we happen to know is a media entity.). We load the media entity and check if there is a field called `field_media_document`. This rather convoluted example is used to get the file size of the file in the media field.

```

$media_id = $node->field_library_media->target_id;
if ($media_id) {
  $media_item = Media::load($media_id);
  // Get the file.
  if ($media_item->hasField('field_media_document')) {
    $file_id = $media_item->field_media_document->getValue()[0]['target_id'];
  }
  if (isset($file_id)) {
    $file = File::load($file_id);
    if ($file) {
      // Get file size.
      $file_size = format_size($file->getSize());
      // Set file size variable.
      $variables['file_size'] = $file_size;
    }
  }
}

```

Create a node and write it to the database

```

use \Drupal\node\Entity\Node;
// Create node object.
$node = Node::create([
  'type' => 'article',
  'title' => 'Druplicon test',
]);
$node->save();

```

Create a node with an image

```

use \Drupal\node\Entity\Node;
use \Drupal\file\Entity\File;

// Create file object from remote URL.
$data = file_get_contents('https://www.drupal.org/files/druplicon.small_.png');
$file = file_save_data($data, 'public://druplicon.png', FILE_EXISTS_REPLACE);

// Create node object with attached file.
$node = Node::create([
  'type' => 'article',
  'title' => 'Druplicon test',
  'field_image' => [
    'target_id' => $file->id(),
    'alt' => 'Hello world',
    'title' => 'Goodbye world'
  ],
]);
$node->save();

```

Write a node with an attached file

```

use \Drupal\node\Entity\Node;
use \Drupal\file\Entity\File;

// Create file object from remote URL.
$data = file_get_contents('https://www.drupal.org/files/druplicon.small_.png');
$file = file_save_data($data, 'public://druplicon.png', FILE_EXISTS_REPLACE);

// Create node object with attached file.
$node = Node::create([
  'type' => 'article',
  'title' => 'Druplicon test',
  'field_image' => [
    'target_id' => $file->id(),
    'alt' => 'Hello world',
    'title' => 'Goodbye world'
  ],
]);
$node->save();

```

To populate the fields of an entity you can either use the `$entity->set($key, $value)` method on the entity object or pass a key=>value array to the entity constructor. As such:

```

$foo = new Foo([
  'name' => 'bar',
  'baz' => TRUE,
  'multi_value' => [
    'first',
    'second',
    'third',
  ]
]);

```

Write a date or datetime to a node

note the [] array stuff is optional

```

//DateTime:
$dateTime = \DateTime::createFromFormat('Y-m-d', '2000-01-30');
$newDateString = $dateTime->format('Y-m-d\TH:i:s');

// Create node object
$node = Node::create([
  'type' => 'workshop',
  'title' => 'selwyn test' . (string) random_int(1,200),
  'field_workshop_start_date' => ['2017-12-05'], //date only
  'field_workshop_end_date' => [$newDateString], //DateTime
]);

```

Or just a date (no time)

```

$dateTime = \DateTime::createFromFormat('Y-m-d', '2000-01-30');
$newDateString = $dateTime->format('Y-m-d');
...
'field_workshop_other_date' => [$newDateString],

```

Set field values

```
$node->set('field_tks_subchapter', $subchapterFullStatement);
$node->set('field_tks_standard_type', "TEKS");
$node->save();
```

And the shortened version (from <https://gorannikolovski.com/blog/various-ways-updating-field-values-drupal-8-and-9>)

```
$node->field_tks_subchapter = $subchapterFullStatement;
$node->field_tks_standard_type = "TEKS";
$node->save();
```

Also

```
$node->field_tks_subchapter->value = $subchapterFullStatement;
$node->field_tks_standard_type->value = "TEKS";
$node->save();
```

Set an entity reference field

```
//That is the node id.
$node->field_name->target_id = 123;

// You can assign an entity object.
$entity->set('field_name', $another_entity);

$node->field_tks_standard_type->value = "TEKS";
$node->save();
```

Set multivalue fields (regular and entity reference)

```
//The multivalue fields are no different. You just have to use arrays. So, instead of this:
$entity->field_name_muti->value = ['foo', 'bar', 'baz'];
$entity->field_name_multi->target_id = [1, 2, 3]
$entity->field_name_multi->target_id = [$another_entity1, $another_entity2, $another_entity3]

// You can use this:

$entity->field_name_muti = ['foo', 'bar', 'baz'];
$entity->field_name_multi = [1, 2, 3]
$entity->field_name_multi = [$another_entity1, $another_entity2, $another_entity3]
```

Also if you just want to update a specific one, specify the delta like this

```
$expectation_node->set('field_name_multi',[0 => 'foo']);
$expectation_node->set('field_name_multi',[1 => 'bar']);
```

Or with variables...

```
$expectation_node->set('field_srp_teacher_voting_status',[${current_vote_number} => ${voting_status}]);
```

Clear a text field

Single text field

```
$node->set('field_subtitle', NULL);
```

Set or clear a body field

```
$node->set('body', [
  'summary' => $summary,
  'value' => $body,
  'format' => 'links_bullets_headings_and_images',
]);
});
```

To empty a body field, use this:

```
$this_node->body[$this_node->language] = [];
```

Load a node and retrieve an entity reference node and nid (target_id)

```
$node_storage = \Drupal::entityTypeManager()->getStorage('node');
$node = $node_storage->load($nid);
//Same
$ref_nid = $node->get('field_sf_contract_ref')->target_id;
//or
$ref_nid = $node->field_sf_contract_ref->target_id;
```

Load a multivalue reference field.

Here node 35 has a `field_event` (multivalue entity reference field) with several values. This code shows how to retrieve the first one, get it's nid and it's title:

```
$node = Node::load(35);
// Get an array of nodes.
$node_array = $node->get('field_event')->referencedEntities();
// Pop one off with reset.
$ref_node = reset($node_array);
$nid = $ref_node->id();
$title = $ref_node->getTitle();
```

As `$ref_node` is a node object, you retrieve any field value with `get()` e.g. `$val = $ref_node->get('field_status')->value;`

You can also loop thru the referenced entities with:

```
foreach ($node_array as $node) {
  $nid = $node->id();
}
```

Entity reference nodes and their fields

You can look at the referenced node's fields but you have to be aware of how you set the display mode.

```
foreach ($node->field_my_entity_reference as $reference) {

  // if you chose "Entity ID" as the display mode for the entity reference field,
  // the target_id is the ONLY value you will have access to
  echo $reference->target_id; // 1 (a node's nid)

  // if you chose "Rendered Entity" as the display mode, you'll be able to
  // access the rest of the node's data.
  echo $reference->entity->title->value; // "Moby Dick"
}
```

Often, you've just loaded the node with `Node::load()` so you have a node object and refer to reference fields like:

```
$vendor_url = $node->field_sf_contract_ref->entity->field_vendor_url->value;
```

Load the taxonomy terms from a term reference field

The referencedEntities returns an array of term objects, so no need to `load()` them separately.

```
if ($node) {
  $topics = $node->get('field_ref_tax')->referencedEntities();
  foreach ($topics as $topic) {
    $term_name = $topic->getName();
  }
}
```

Load a node and find the terms referenced in a paragraph in a term reference field

Here we loop thru all the instances of my paragraph reference and grab the term in the paragraph.

```
foreach ($node->get('field_my_para')->referencedEntities() as $ent){
  $term = $ent->$field_in_paragraph->entity;
  $name = $term->getName();
  print_r($name);
}
```

Retrieve a URL field

External links

You can get the URL (for external links) and then just the text part. Note this doesn't work for internal links. Note also this slightly convoluted example has a reference field `field_sf_contract_ref` which has a link to another entity and the `field_vendor_url->first()->getUrl()` is the important part. Also note, this is a single-value field (not a multivalue field – so the `first()` call is a little disturbing)

```
$vendor_url = $node->field_sf_contract_ref->entity->field_vendor_url->first()->getUrl();
if ($vendor_url) {
  $vendor_url = $vendor_url->getUri();
  //OR
  $vendor_url = $vendor_url->toString();
}
```

A slightly simpler example from `modules/custom/tea_teks/modules/tea_teks_srp/src/Form/SrpAddEditCitationForm.php`

```
$citation_link = $citation->get('field_link');
if (!$citation_link->isEmpty()) {
  $citation_link = $citation->field_link->first()->getUrl()->toString();
}
```

Internal links

For internal links, use `getUrl()` for the URL and `->title` for the title.

```

$instructions_node = Node::load($order_type_instructions_nid);

if ($instructions_node) {
  $order_link = $instructions_node->field_link->first();

  if ($order_link) {
    $uri = $order_link->uri;
    $variables['order_link_title'] = $order_link->title;
    $order_url = $order_link->getUrl();
    if ($order_url) {
      $variables['order_type_link'] = $order_url;
    }
  }
}

```

Load a node and retrieve a paragraph field

Because paragraphs and nodes are both entities, the pattern is the same. You load the entity (node or paragraph) and then simply reference the field name e.g. myentity->field_blah

From /Users/selwyn/Sites/inside-mathematics/themes/custom/danaprime/danaprime.theme

These are a little different from regular fields. Generally you want to get their target_id which will tell you the pid or paragraph id. Here are two different ways to load a video_collection_node and go to retrieve a field field_related_lessons which holds paragraphs of type related_lessons

```

$video_collection_node = Node::load($video_collection_nid);

//This gives you a bunch of \Drupal\entity_reference_revisions\EntityReferenceRevisionsFieldItemList items
$lessons = $video_collection_node->field_related_lessons;
//or
$lessons = $video_collection_node->get('field_related_lessons');

foreach ($lessons as $lesson) {
  $paragraph_revision_ids[] = $lesson->target_revision_id;
}

```

Paragraphs use the contributed Entity Reference Revisions module to reference paragraphs and it is very important to use the `target_revision_id` property when referencing paragraphs. Alternatively, the `entity` computed property can be used to retrieve the paragraph entity itself.

OR

```

//This gives you an array of arrays [[target_id => '348', target_revision_id => '348'], [target_id => '349', target_revision_id => '349']]
$lessons = $video_collection_node->get('field_related_lessons')->getValue();
foreach ($lessons as $lesson) {
  $paragraph_revision_ids[] = $lesson['target_revision_id'];
}

```

Collecting them like this is only an example, while the `loadMultiple` method exists on entity storage objects, there is no `loadMultipleRevisions` method.

EH!

```

// This gives you null! - don't do this.
$lessons = $video_collection_node->get('field_related_lessons')->value;

```

Note. getValue() here will get you the nid buried in a result array of arrays like result[0]['target_revision_id'] - quicker to just grab ->target_revision_id

```

## Load a node and grab a paragraph field to find the nid in an entity reference field

From
/Users/selwyn/Sites/inside-mathematics/themes/custom/danaprime/danaprime.theme -
Continuing from above, I load a node, grab it's field
`field_related_lessons` which holds paragraphs of type `related_lessons`
and grab it's field `field_lesson.` That field has a target_id which is
the nid for the entity reference field. Phew!

```php
//Grab the related lessons from the collection.
$video_collection_node = Node::load($video_collection_nid);
$lessons = $video_collection_node->field_related_lessons;
$storage = \Drupal::entityTypeManager()->getStorage('paragraph');
foreach ($lessons as $lesson) {
 //Load each paragraph and get the nids from them.
 $paragraph = $lesson->entity;
 $related_lessons_nid = $paragraph->field_lesson->target_id;
 $related_lessons_nids[] = $related_lessons_nid;
}

//This should have an array of nids for video_details.
$variables['related_lessons_pids'] = $pids;
$variables['related_lessons_nids'] = $related_lessons_nids;

```

## How to get Node URL alias or Taxonomy Alias by Node id or Term ID

Drupal 8 uses `url` objects to easily generate and manipulate paths. The easiest way is to load the entity and convert it to a URL object:

```
$node = Node::load(1234)
$url_object = $node->toUrl();
```

This now can be converted to a string

```
$href = $url_object->toString();
```

If an absolute URL is needed, that's easy too:

```
$href = $url_object->setAbsolute()->toString();
```

This will return a path in the form `node/123`

```
$current_path = \Drupal::service('path.current')->getPath();
```

## How to set a URL Alias

Since Drupal 9, url aliases are entities so:

From: /Users/selwyn/Sites/dir/web/modules/custom/dir/dir.module

```

$node_path = "/node/$nid";
$new_url = $parent_url_alias . $url_alias;

/** @var \Drupal\path_alias\PathAliasInterface $path_alias */
$path_alias = \Drupal::entityTypeManager()->getStorage('path_alias')->create([
 'path' => $node_path,
 'alias' => $new_url,
 'langcode' => 'en',
]);
$path_alias->save();

```

## Get a node's menu item and more

Here we get the current route's menu item using it's nid, reset the link off the array and we can extract the URL and other exciting things. Mostly you want to check its children, parents etc. Here we grab it's URL..

```

/** @var \Drupal\Core\Menu\MenuLinkManagerInterface $menu_link_manager */
$menu_link_manager = \Drupal::service('plugin.manager.menu.link');
$links = $menu_link_manager->loadLinksByRoute('entity.node.canonical', ['node' => $nid]);
$link = reset($links);
$urlobject = $link->getUrlObject();
$url_string = $urlobject->toString();
$x = $link->getParent(); //get the parent menu item GUID.
$y = $link->getTitle(); //get the title of the menu item.

$menu_name = $link->getMenuName(); //get the menu name e.g. "main"

```

## Find a node using it's uuid

```

/**
 * Tries to get node for given uuid if it exists.
 *
 * @param string $uuid
 * UUID to search for existing node.
 *
 * @return \Drupal\Core\Entity\EntityInterface[]|null
 * Node object or NULL.
 *
 * @throws \Drupal\Component\Plugin\Exception\InvalidPluginDefinitionException
 * @throws \Drupal\Component\Plugin\Exception\PluginNotFoundException
 */
public function getExistingUpdateNode(string $uuid) {
 try {
 $existing_node = \Drupal::entityTypeManager()
 ->getStorage('node')
 ->loadByProperties(['uuid' => $uuid]);
 return $existing_node;
 } catch (RequestException $e) {
 watchdog_exception('ncs_infoconnect', $e);
 return NULL;
 }
}

```

## Retrieve Node ID(NID) or Taxonomy term ID from a Drupal alias or path

If alias-path does not exist, it will return the same argumented string.

The first example will work for the node, second for the taxonomy and third for the users. For Drupal 8.8.0 and later use

## path\_alias.manager

```
//Node
$path = \Drupal::service('path.alias_manager')->getPathByAlias('/alias-path');
if (preg_match('/node/(\d+)/', $path, $matches))
{
 $nodeID = $matches[1];

//Taxonomy term
$path = \Drupal::service('path.alias_manager')->getPathByAlias('/alias-path');
if (preg_match('/taxonomy/term/(\d+)/', $path, $matches))
{
 $termID = $matches[1];
}

//User
$path = \Drupal::service('path.alias_manager')->getPathByAlias('/alias-path');
if (preg_match('/user/(\d+)/', $path, $matches))
{
 $userID = $matches[1];
}
```

## Retrieve all nodes with a matching taxonomy term

```
$nodes = \Drupal::entityTypeManager()->getStorage('node')->loadByProperties([
 'field_tags' => $term_id,
]);
```

## Load first 5 nodes that have a matching taxonomy term

```
```php
protected function loadFirstOpinion($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->range(0, 5);
  $nids = $query->execute();
  $nodes = $storage->loadMultiple($nids);

  $ra = [];
  foreach ($nodes as $node) {
    $ra[] = [
      '#type' => 'markup',
      '#markup' => '<p>' . $node->getTitle(),
    ];
  }
  return $ra;
}
```

How to uncache a particular page or node

This will cause Drupal to rebuild the page internally, but won't stop browsers or CDN's from caching.

```
\Drupal::service('page_cache_kill_switch')->trigger();
```

You can use this statement in node_preprocess, controller, etc.

You will need a custom module to implement setting max-age to 0 like this. In a .module file:

```

use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Entity\Display\EntityViewDisplayInterface;

function drt_node_view_alter(array &$amp;build, EntityInterface $entity, EntityViewDisplayInterface $display) {
  $bundle = $entity->bundle();
  if ($bundle == 'search_home') {
    $build['#cache']['max-age'] = 0;
    \Drupal::service('page_cache_kill_switch')->trigger();
  }
}

```

Note. The above custom module is called drt. You would use your own module name instead of drt when naming the function.

Get boolean Field

Boolean fields show up as 0 or 1 so a simple test using if \$val will return true for yes/on and false for no/off.

```

$val = $contract_node->field_erate_certification->value;
if ($val) {
  $erate_certification = "Yes";
}

```

Date Field

Date fields in Drupal are stored in UTC date strings. When you load them, you have some options. Probably grabbing the field->date is the best so you can use all the goodness of the DrupalDateTime class. If you need to do calculations involving unix timestamps, then ->getTimestamp is useful although DrupalDateTime can do all kinds of calculations too.

```

$end_date = $contract_node->field_contract_end_date->value;
$end_date = $contract_node->field_contract_end_date->date->getTimestamp();
$end_date = $contract_node->field_contract_end_date->date;
$formatted_date = $end_date->format('m/d/y');

field_contract_end_date->value; //returns whatever the string is e.g. 2024-08-31
field_contract_end_date->date->getTimestamp(); // returns unix timestamp e.g. 1725105600
field_contract_end_date->date; // returns a DrupalDateTime object with all it's goodness

$formatted_date = $end_date->format('m/d/y'); //Format the date nicely for output

```

From <https://drupal.stackexchange.com/questions/252333/how-to-get-formatted-date-string-from-a-datetimeitem-object>

A date field has two properties,

- value to store the date in UTC and
- date, a computed field returning a DrupalDateTime object, on which you can use the methods getTimestamp() or format():

```

// get unix timestamp
$timestamp = $node->field_date->date->getTimestamp();
// get a formatted date
$date_formated = $node->field_date->date->format('Y-m-d H:i:s');

```

Date Fields

See also Queries: "Query the creation date (among other things) using entityQuery"

These are stored in the Drupal database as varchar 20 strings.

```
$d = date("m/d/Y", $timestamp);
```

printing this date will return: 08/16/2018

From <https://www.drupal.org/node/1834108>

There is a DrupalDateTime object which is used like this:

```
$date = DrupalDateTime::createFromDate($datetime);
$date = DrupalDateTime::createFromArray($array);
$date = DrupalDateTime::createFromTimestamp($timestamp);
$date = DrupalDateTime::createFromFormat($format, $time);
```

e.g. to create a DrupalDateTime

```
$format = 'Y-m-d H:i';
$start_date = DrupalDateTime::createFromFormat($format, "2019-01-01 00:00");
```

In the database the created and changed fields use a unix timestamp. e.g look in node_field_revision

This is an int 11 field in the db with a value like 1525302749 (Note. Negative values are dates before 1970 or epoch which require some slightly special magic)

If you add a Date field to a content type, it's data looks like 2019-05-15T21:32:00 (varchar 20)

See the next item below for how to query the created and changed fields and also the fun you have to go through to query date fields.

When writing Drupal dates to the database, you need to go thru some machinations. Here are some details explain the timezone.

```
$date_string = "2020-08-24T15:28:04+00:00";
$tz = \Drupal::currentUser()->getTimezone();
// specify timezone as "America/Chicago"
$given = new \Drupal\Core\Datetime\DrupalDateTime($date_string, $tz);
// specify timezone as "UTC"
$given = new \Drupal\Core\Datetime\DrupalDateTime($date_string, "UTC");
// specify UTC as the timezone
$given->setTimezone(new \DateTimeZone("UTC"));
// The date string appears adjusted to UTC
$newstring = $given->format("Y-m-d\Th:i:s");
```

Date Range Field doesn't display correct timezone

If you use a date range field in your content type and try to display the time part in a template, Drupal will cleverly display the UTC version (*not* your current timezone version) so the time will be off by potentially ~6 hours.

Using the field field_event_date, here is the twig template code to display the date. You'd expect it to show the correct timezone

```
{{ node.field_event_date.0.value|date('g:i:a') }} - {{ node.field_event_date.0.end_value|date('g:i:a') }}
```

If you simply used {{ content }} all fields are displayed correctly – timezone is correct.

If you use {{ content.field_date_start }} – timezone show correctly also

But I want to grab the time only to display separately from the date and put this in the twig template:

```
{{ node.field_date_start.value|date("g:i:a") }}
```

It fails.

From

The solution is to create a variable in the template_preprocess_node. Append 'UTC' to the datetime string and make a timestamp so you get the UTC time (which matches what is in the Drupal DB.). Grab the user's timezone for conversion purposes. Use the Drupal date.formatter service to format a timestamp (using the timezone) and you are good to go.

```

function txglobal_preprocess_node(&$variables) {
  ..

  if ( ($node_type == 'event') ) {

    $node = $variables['node'];
    $timezone = drupal_get_user_timezone();
    $formatter = \Drupal::service('date.formatter');

    $dates = $node->get('field_event_date')->getValue();

    $date_start = $dates[0]['value'];
    $time = strtotime($date_start.' UTC');
    $start_time = $formatter->format($time, 'custom', 'g:ia', $timezone);
    // $start_time = date("g:ia", $time);

    $date_end = $dates[0]['end_value'];
    $time = strtotime($date_end.' UTC');
    $end_time = $formatter->format($time, 'custom', 'g:ia', $timezone);

    // $date_start = new DateTime($date_start);
    // $timestamp_start = $date_start->getTimestamp();
    // $start_time = $formatter->format($timestamp_start, 'custom', 'g:ia', $timezone);
    // $end_time = $formatter->format($date_end, 'custom', 'g:ia', $timezone);

    $variables['start_time'] = $start_time;
    $variables['end_time'] = $end_time;
  }
}

```

and then in the template, use

`_`

Interestingly, if you use the [smart_date module](#), you might use this version. Notice that dates are already stored as timestamps so you don't have to first convert them.

```

if ( $node_type == 'event' && $view_mode == 'default' ) {
  $timezone = drupal_get_user_timezone();
  $formatter = \Drupal::service('date.formatter');

  $d = $node->get('field_smart_event_date')->getValue();
  $d0s = $d[0]['value']; //first date start time
  $d0e = $d[0]['end_value']; //first date end time
  $start_time = $formatter->format($d0s, 'custom', 'g:ia');
  $end_time = $formatter->format($d0e, 'custom', 'g:ia', $timezone);
  $variables['start_time'] = $start_time;
  $variables['end_time'] = $end_time;
}

```

Date Range

```

// formatted start date
$start_date_formatted = $node->field_date->start_date->format('Y-m-d H:i:s');
// formatted end date
$end_date_formatted = $node->field_date->end_date->format('Y-m-d H:i:s');

```

Date Range fields: Load start and end values

Start date and end date.

```
$node->get('field_cn_start_end_dates')->value  
$node->get('field_cn_start_end_dates')->end_value
```

Date Fields: Load or save them

Because date fields in Drupal are stored in strings, when you use `get()` or `set()`, they are still strings.

```
$node->get('field_cn_start_end_dates')->value  
  
$node->set('field_cn_end_date', $end_date);
```

If you want to manipulate them, convert them to `DrupalDateTime` objects, then convert them back to strings

Creating a new `DrupalDateTime` object is done like this:

```
use Drupal\Core\Datetime\DrupalDateTime;  
  
$date = DrupalDateTime::createFromFormat('j-M-Y', '20-Jul-2019');  
  
$date = new DrupalDateTime('now'); // grab current dateTime using NON static  
$date->format('l, F j, Y - H:i'); // format it  
// prints out nicely formatted version: Tue, Jul 16, 2019 - 11:34:am  
  
$date = new DrupalDateTime('now'); // grab current dateTime  
// Or print $date->format('d-m-Y: H:i A');  
// prints out: 16-07-2019: 11:43 AM
```

How about with timezones?

```
$date = new DrupalDateTime();  
$date->setTimezone(new \DateTimeZone('America/Chicago'));  
print $date->format('m/d/Y g:i a');  
// The above prints current time for given Timezone  
// prints : 07/16/2019 10:59 am  
  
// Another variations of the above except it takes specific date and UTC zone  
$date = new DrupalDateTime('2019-07-31 11:30:00', 'UTC');  
$date->setTimezone(new \DateTimeZone('America/Chicago'));  
print $date->format('m/d/Y g:i a');  
// prints 07/31/2019 6:30 am
```

The code below shows adding `$days` (an integer) to the date value retrieved from the field: `field_cn_start_date`.

Don't forget to add the `use` statement.

```
use Drupal\Core\Datetime\DrupalDateTime;  
  
$start_date_val = $node->get('field_cn_start_date')->value;  
$days = intval($node->get('field_cn_suspension_length')->value) - 1;  
  
// $end_date = DrupalDateTime::createFromFormat('Y-m-d H:i:s', $start_date_val . " 00:00:00");  
$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val );  
$end_date->modify("+$days days");  
$end_date = $end_date->format("Y-m-d");  
  
$node->set('field_cn_end_date', $end_date);
```

Comparing DrupalDateTime values

DrupalDateTimes are derived from DateTimePlus which are wrappers for PHP `DateTime` class. So you can use that functionality to do comparisons

```
date_default_timezone_set('Europe/London');

$d1 = new DateTime('2008-08-03 14:52:10');
$d2 = new DateTime('2008-01-03 11:11:10');
var_dump($d1 == $d2);
var_dump($d1 > $d2);
var_dump($d1 < $d2);
```

outputs:

```
bool(false)
bool(true)
bool(false)
```

Date with embedded timezone

Here you have a date string with an embedded timezone so you can make it into a `DrupalDateTime` and then format it into a usable string you can store in the db.

```
use Drupal\Core\Datetime\DrupalDateTime;

$date_string = "2020-08-24T15:28:04+00:00";
$given = new DrupalDateTime($date_string);
$newstring = $given->format("Y-m-d\Th:i:s");
```

Has something expired?

Checking to see if the `expiration_date` has passed. The `field_expiration_date` is a standard Drupal date field in the `sf_resellers` content type.

```
$source_node = $node_storage->load($nid);

$expiration_date = $source_node->field_expiration_date->value;

// Use expiration date to un-publish expired resellers to hide them.
$status = 1;
if ($expiration_date) {
  $expirationDate = DrupalDateTime::createFromFormat('Y-m-d', $expiration_date);
  $now = new DrupalDateTime();
  if ($expiration_date < $now) {
    $status = 0;
  }
}
```

TODO: date fields are stored in UTC. Need to factor in timezone.

Load or save Drupal Date fields

Date fields in Drupal are stored in strings

If you want to load them, just `get()`, as below and to save them, use `set`.

```
$node->get('field_cn_start_end_dates')->value

$node->set('field_cn_end_date', $end_date);
```

If you want to manipulate them, convert them to `DrupalDateTime` objects, then convert them back to strings like this. `$days` here is an int and we are adding that to the date value. Don't forget to add the use statement.

```
use Drupal\Core\Datetime\DrupalDateTime;

$start_date_val = $node->get('field_cn_start_date')->value;
$days = intval($node->get('field_cn_suspension_length')->value) - 1;

// $end_date = DrupalDateTime::createFromFormat('Y-m-d H:i:s', $start_date_val . " 00:00:00");
$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val );
$end_date->modify("+$days days");
$end_date = $end_date->format("Y-m-d");

$node->set('field_cn_end_date', $end_date);
```

Here you have a slightly funny date string, you make it into a `DrupalDateTime` and then format it into a usable string you can store in the database.

```
use Drupal\Core\Datetime\DrupalDateTime;

$date_string = "2020-08-24T15:28:04+00:00";
$given = new \Drupal\Core\Datetime\DrupalDateTime($date_string);
$newstring = $given->format("Y-m-d\Th:i:s");
```

Retrieve node creation date and format it

Here I query the database for a node and return a string version of the date. It probably makes more sense to return the date and do the formatting at the theme level. Note. This will handle epoch dates before 1970.

```
use Drupal\Core\Datetime\DrupalDateTime;

protected function loadFirstOpinionYear($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') //DESC
    ->range(0, 1);
  $nids = $query->execute();
  if ($nids) {
    $node = $storage->load(reset($nids));
  }
  $time = $node->get('created')->value;

  $d = new \DrupalDateTime("@$time"); //can use either this
  $d = new \DateTime("@$time"); // or this..

  $str = $d->format('Y-m-d H:i:s');
  return $str;
}
```

Retrieve node creation or changed date and format it

Both created and changed are stored as Unix epoch timestamps the `node_field_data` table. Here I query the database for a node and return a string version of the date. Note. This will handle epoch dates before 1970.

```

use Drupal\Core\Datetime\DrupalDateTime;

protected function loadFirstOpinionYear($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') //DESC
    ->range(0, 1);
  $nids = $query->execute();
  if ($nids) {
    $node = $storage->load(reset($nids));
  }
  $time = $node->get('created')->value;

  $d = new DrupalDateTime("@$time"); //can use either this
  $d = new \DateTime("@$time"); // or this..

  $str = $d->format('Y-m-d H:i:s');
  return $str;
}

```

Date Field and Date with no time (remove time)

Use the `setTime()` function to remove the time part of a datetime so we can make comparisons of just the date.

```

function ogg_mods_cn_form_validate($form, FormStateInterface $form_state) {
  $start_date = $form_state->getValue('field_cn_start_date');
  if ($start_date) {
    $start_date = $start_date[0]['value'];
    $start_date->setTime(0, 0, 0);
    $now = new Drupal\Core\Datetime\DrupalDateTime();
    $now->modify("-2 days");
    $now->setTime(0, 0, 0);

    \Drupal::messenger()->addMessage("Start date = $start_date");
    \Drupal::messenger()->addMessage("Now date - 2 days = $now");

    if ($start_date < $now) {
      $form_state->setErrorByName('edit-field-cn-start-date-0-value-date', t('The starting date is more than 2 days in the past. Please select a later date'));
    }
  }
}

```

Smart date (smart_date) load and format

Load the smart date field and use the drupal date formatting service. Smart date fields are always stored as unix timestamp values e.g. 1608566400 which need conversion for human consumption.

```
$start = $node->field_when->value;
$formatter = \Drupal::service('date.formatter');
$start_time = $formatter->format($start, 'custom', 'm/d/Y g:ia'); //12/21/2020 10:00 am
```

Alternatively, you could load it, create a `DrupalDateTime` and then format it

```
``php
$start = $node->field_when->value;
$dtt = DrupalDateTime::createFromTimestamp($start);
$start_date = $dtt->format('m/d/Y'); //returns 12/21/20
$start_time = $dtt->format('g:ia'); // returns 10:00am
```

Smart date (smart_date) all-day

To check if a smart date is set to all day, check the duration. If it is 1439, that means all day.

```
$start_ts = $node->field_when->value;
$start_dt = DrupalDateTime::createFromTimestamp($start_ts);
$start_date = $start_dt->format('m/d/Y');
$duration = $node->field_when->duration; //1439 = all day

if ($duration == 1439) {
  $start_time = "all day";
}
else {
  $start_time = $start_dt->format('g:ia');
}
```

Smart date (smart_date) range of values

```
//Event start date.
$whens = $node->get('field_when'); //produces SmartDateFieldItemList

// Each $when is a \Drupal\smart_date\Plugin\Field\FieldType\SmartDateItem
foreach ($whens as $when) {
  $start = $when->value;
  $end = $when->end_value;
  $duration = $when->duration; //1439 = all day
  $tz = $when->timezone; //"" means default. Uses America/Chicago type format.
}
```

You can also peek into the repeating rule and repeating rule index. These are in the `smart_date_rule` table and I believe the index identifies which item is in the `instances` column.

```
$rrule = $when->rrule;
$rrule_index = $when->rrule_index;
```

hook_node_presave or hook_entity_type_presave

When you want to fiddle with a node as it is being saved, use `hook_node_presave()`

Note there is some good date arithmetic here

```

/**
 * Implements hook_ENTITY_TYPE_presave().
 */
function ogg_mods_node_presave(NodeInterface $node) {
  switch ($node->getType()) {
    case 'catastrophe_notice':
      $end_date = NULL != $node->get('field_cn_start_end_dates')->end_value ? $node->get('field_cn_start_end_dates')->end_value : 'n/a';
      $govt_body = NULL != $node->field_cn_governmental_body->value ? $node->field_cn_governmental_body->value : 'Unnamed Government Body';
      $start_date_val = $node->get('field_cn_start_date')->value;

      $accountProxy = \Drupal::currentUser();
      $account = $accountProxy->getAccount();
      // Anonymous users automatically fill out the end_date.
      if (!$account->hasPermission('administer catastrophe notice')) {
        $days = intval($node->get('field_cn_suspension_length')->value) - 1;

        $end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
        $end_date->modify("+$days days");
        $end_date = $end_date->format("Y-m-d");
        $node->set('field_cn_end_date', $end_date);
      }

      // Always reset the title.
      $title = substr($govt_body, 0, 200) . " - $start_date_val";
      $node->setTitle($title);

    /*
     * Fill in Initial start and end dates if this is an extension of
     * a previously submitted notice.
     */
      $extension = $node->get('field_cn_extension')->value;
      if ($extension) {
        $previous_notice_nid = $node->get('field_cn_original_notice')->target_id;
        $previous_notice = Node::load($previous_notice_nid);
        if ($previous_notice) {
          $initial_start = $previous_notice->get('field_cn_start_date')->value;
          $initial_end = $previous_notice->get('field_cn_end_date')->value;
          $node->set('field_cn_initial_start_date', $initial_start);
          $node->set('field_cn_initial_end_date', $initial_end);
        }
      }
      break;
  }
}

```

Disable caching for a content type

If someone tries to view a node of content type `search_home` (entity of bundle `search_home`) caching is disabled and Drupal and the browser will always re-render the page. This is necessary for a page that is retrieving data from a third party source and you almost always expect it to be different. It wouldn't work for a search page to show results from a previous search.

```

use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Entity\Display\EntityViewDisplayInterface;

use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Entity\Display\EntityViewDisplayInterface;

/**
 * Implements hook_ENTITY_TYPE_view_alter().
 */
function dir_node_view_alter(array &$amp;build, EntityInterface $entity, EntityViewDisplayInterface $display) {
  $bundle = $entity->bundle();
  if ($bundle == 'search_home') {
    $build['#cache']['max-age'] = 0;
    \Drupal::service('page_cache_kill_switch')->trigger();
  }
}

```

In the API docs (

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Entity%21entity.api.php/function/hook_ENTITY_TYPE_view_alter/9.2.x

) They have an example where they check the view mode and dynamically add a field to the node. There is also a post_render callback function added to do some more magic. To implement, you'd have to replace ENTITY_TYPE with your entity type such as node. See below:

```

function hook_ENTITY_TYPE_view_alter(array &$build, Drupal\Core\Entity\EntityInterface $entity, \Drupal\Core\Entity\Display\EntityViewDisplayInterface $display) {
  if ($build['#view_mode'] == 'full' && !isset($build['an_additional_field'])) {

    // Change its weight.
    $build['an_additional_field']['#weight'] = -10;

    // Add a #post_render callback to act on the rendered HTML of the entity.
    $build['#post_render'][] = 'my_module_node_post_render';
  }
}

```

Writing some JSON data into a long text field

It is sometimes useful to store JSON data into a long text field. I've used it for storing formatted status information about a complex process.

```

$aggregate_values = [
  'citation_status' => $citation_status,
  'accepted_votes' => $accepted_votes,
  'rejected_votes' => $rejected_votes,
];
$aggregate_data = Json::encode($aggregate_values);
$citation_node->field_srp_agg_data_json[$vote_number] = $aggregate_data;
$citation_node->save();

```

The JSON data is written into a multivalue long text field using the \$vote_number index. So if \$vote_number is 0, this will be the first value. If it is 1, then this will be written into the second value. This is what it looks like in Drupal on a node edit screen.

SRP: AGGREGATED DATA

```
{"citation_status": "incomplete", "accepted_votes": 0, "rejected_votes": "1"}
```



Stores aggregated data for this content type, for quick historical retrieval. JSON encoded array data. Index of field should match srp_vote_number. citation_status: [status] accepted_votes: [number] rejected_votes: [number] revision_note: [text]

[Add another item](#)

Create a node with an image

```
use \Drupal\node\Entity\Node;
use \Drupal\file\Entity\File;

// Create file object from remote URL.
$data = file_get_contents('https://www.drupal.org/files/druplicon.small_.png');
$file = file_save_data($data, 'public://druplicon.png', FILE_EXISTS_REPLACE);

// Create node object with attached file.
$node = Node::create([
  'type'      => 'article',
  'title'     => 'Druplicon test',
  'field_image' => [
    'target_id' => $file->id(),
    'alt' => 'Hello world',
    'title' => 'Goodbye world'
  ],
]);
$node->save();
```

Paragraphs

Paragraphs are those special things that allow you to blend a couple of field together e.g. count and unit of measure so you can store values like 5 kilograms or 7 years etc. Often they are used like nodes where you define the fields and fill them with data that get displayed on the screen for things like carousels or events.

Load a node and find the terms referenced in a paragraph in a term reference field

Here we loop thru all the instances of my paragraph reference and grab the term in the paragraph.

```
use Drupal\taxonomy\Entity\Term;

foreach ($node->get('field_my_para')->referencedEntities() as $ent){
  $term = Term::load($ent->$field_in_paragraph->target_id);
  $name = $term->getName();
  print_r($name);
}
```

Custom Field Formatter

Custom field formatters can be used in display modes or in views. These are very powerful.

Some basic info is available at <https://www.drupal.org/docs/8/creating-custom-modules/create-a-custom-field-formatter>

This example is a custom formatter that takes a value from a field (in this case a uuid) and builds a url which essentially retrieves an image (via an API call.) It looks for some config info (in the node display mode for the node, or in the views setup for the usage in a view.).

For the node called infofeed, the config data is stored in an entity called core.entity_view_display.node.infofeed.default

For a view called infofeeds, the config data is stored in a config entity called views.view.infofeeds.

(You can find them by browsing thru the config table and looking for your info in the data field i.e. in sequel pro, look for data like %image_width%)

It is pretty reasonable that the custom field formatter will require some configuration, so this means we will need a module/config/schema/module.schema.yml file

So at /Users/selwyn/Sites/ncs/docroot/modules/custom/ncs_infoconnect/config/schema/ncs_infoconnect.schema.yml we have the following file which defines a config_entity called NCS Thumbnail settings and specifically two integer values for image_width and image height. I use these to specify the size of the thumbnail I generate:

```
# Schema for configuring NCS thumbnail formatter.
```

```
field.formatter.settings.ncs_thumbnail:
  type: config_entity
  label: 'NCS thumbnail settings'
  mapping:
    image_width:
      label: 'Image width'
      type: integer
    image_height:
      label: 'Image Height'
      type: integer
```

I create the fieldformatter as a fairly unexciting plugin at
/Users/selwyn/Sites/ncs/docroot/modules/custom/ncs_infoconnect/src/Plugin/Field/FieldFormatter/NcsThumbnailFormatter.php

The annotation shows what will be seen in Drupal when configuring the formatter.

```
<?php

namespace Drupal\ncs_infoconnect\Plugin\Field\FieldFormatter;

use Drupal\Core\Field\FormatterBase;
use Drupal\Core\Field\FieldItemListInterface;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Site\Settings;

/**
 * Plugin implementation of the 'ncs_thumbnail' formatter.
 *
 * @FieldFormatter(
 *   id = "ncs_thumbnail",
 *   label = @Translation("NCS Thumbnail"),
 *   field_types = {
 *     "string"
 *   }
 * )
 */
class NcsThumbnailFormatter extends FormatterBase {
```

I override the settingsSummary() which is mostly informative, and viewElements() which has the meat of the plugin. In viewElements(), we loop thru the

items (i.e. the values coming in from the field) and build an image_uri, jam each one into a render element and return the bunch.

```
/*
 * {@inheritDoc}
 */
public function settingsSummary() {
    $summary = [];
    $summary[] = $this->t('Specify size of the thumbnail to display.');
    return $summary;
}

/*
 * {@inheritDoc}
 */
public function viewElements(FieldItemListInterface $items, $langcode) {
    $elements = [];
    $markup = '';
    $width = $this->getSetting('image_width');
    $height = $this->getSetting('image_height');
    $ncs_auth_settings = Settings::get('ncs_api_auth', []);
    $base_url = $ncs_auth_settings['default']['imageserver'];

    foreach ($items as $delta => $item) {
        $image_uri = $base_url . "?uuid=" . $item->value . "&function=original&type=thumbnail";
        $markup = '';

        // Render each element as markup.
        $elements[$delta] = [
            '#markup' => $markup,
        ];
    }

    return $elements;
}
```

Note. Retrieving the config settings for a particular situation happens with a call to `getSetting()` as in:

```
$width = $this->getSetting('image_width');
$height = $this->getSetting('image_height');
```

To use this we need to edit the display for the `infofeed` content type, make sure we have the `image_uuid` field displayed (i.e. not disabled) for Format, select NCS Thumbnail, click the gear to the right to specify the thumbnail size and save. Displaying nodes will then include the thumbnails.

You can do the same with a view: Add the field, specify the formatter (and dimensions) and the thumbnail will appear.

Puzzles

What can I do with a call to `first()` on an entity reference field?

After loading a node, I want to see the value in an entity reference field. I can call `referencedEntities` to pull out its values and loop thru them – I get Nodes in that instance.

```
$refs = $node_to_update->get('field_sf_account_ref')->referencedEntities();
if ($refs) {
    $ref = reset($refs);
    $node_to_update_sf_account_nid = $ref->id();
```

However, if I call `->first()`, I get an `EntityReferenceItem`. I'm curious how I could use this – would I want to?

```
$refs = $node->get('field_sf_dir_contact_ref')->first(); //returns an EntityReferenceItem.
```

Great Cheat sheets

Various ways of updating field values in Drupal 8 and 9

<https://gorannikolovski.com/blog/various-ways-updating-field-values-drupal-8-and-9>

Entity query cheat sheet:

<https://www.metaload.com/blog/drupal-8-entity-api-cheat-sheet>

Drupal entity API cheat sheet

<https://drupalsun.com/zhilevan/2018/07/21/drupal-entity-api-cheat-sheet>

[Back to top](#)

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

Page last modified: Jun 20 2023.

[Edit this page on GitHub](#)

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

Queries

TABLE OF CONTENTS

- [entityQuery](#)
 - [Find matching nodes - example 1](#)
 - [Find matching nodes - example 2](#)
 - [Find matching article nodes – example 3](#)
 - [Find nodes that match a taxonomy term](#)
 - [Find 5 nodes that have a matching taxonomy term](#)
 - [Find matching nodes and delete them](#)
 - [Slice up entityQuery results into batches of 100 nodes](#)
 - [Query the creation date \(among other things\) using entityQuery](#)
 - [entityQuery frequently used conditions](#)
 - [Update menu items programatically](#)
 - [Query multi-value fields](#)
 - [Query entity reference fields if they have a value or no value](#)
 - [Query entity reference fields](#)
- [Static and dynamic Queries](#)
 - [Static Queries](#)
 - [Get a connection object](#)
 - [SQL select example](#)
 - [Find the biggest value in a field](#)
 - [SQL update query - example 1](#)
 - [SQL update query - example 2](#)
 - [SQL update query - example 3](#)
 - [SQL insert](#)
 - [SQL Insert Query](#)
 - [SQL Delete query](#)
 - [Paragraph query](#)
 - [Create a custom table for your module](#)
- [Reference](#)

views 150

For most work, I use entityQueries. There are a few circumstances where I've needed to get into the SQL which meant using static or dynamic queries. There are examples of these different techniques below.

entityQuery

Find matching nodes - example 1

In this EntityQuery example we search for nodes of content type (bundle) ws_product and match field_product_sku with the \$sku variable.

```
use Drupal\node\Entity\Node;
```

```

function getProductId($sku) {
  $productId = false;
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'ws_product')
    ->condition('field_product_sku', $sku);

  $nids = $query->execute();
  if ($nids) {
    $nid = array_values($nids);
    $node = Node::load($nid[0]);
    $productId = $node->get('field_product_id')->value;
  }
  return $productId;
}

```

Find matching nodes - example 2

In this entityQuery we search for published nodes of type contract with field_contract_status having the value “Active”. This puts the resulting nids and node titles in a render array for display.

This is a simple query which outputs a bunch of nids and titles

```

public function loadRawSalesforceData() {
  $node_storage = \Drupal::entityTypeManager()->getStorage('node');

  $query = \Drupal::entityQuery('node')
    ->condition('type', 'contract')
    ->condition('status', 1)
    ->condition('field_contract_status', 'Active')
    ->sort('title', 'DESC');

  $nids = $query->execute();
  if ($nids) {
    $nodes = $node_storage->loadMultiple($nids);
    foreach ($nodes as $node) {
      $nid = $node->id();
      $titles[] = [
        '#type' => 'markup',
        '#markup' => "<p>" . "nid=$nid " . "Title=" . $node->getTitle(). "</p>",
      ];
    }
    return $titles;
  }
  return [
    '#markup' => $this->t('nothing, nada, not a sausage'),
  ];
}

```

Find matching article nodes – example 3

This example looks for an entity of type ‘article’ with the name \$name

```

public function entityExists() {

  $name = 'hello';
  // See if the article named hello exists.
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'article')
    ->condition('title', $name)
    ->count();

  $count_nodes = $query->execute();

  if ($count_nodes == 0) {
    $str = "Found no articles";
  }
  elseif ($count_nodes > 0) {
    $str = "Found $count_nodes articles";
  }

  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $render_array;
}

```

Find nodes that match a taxonomy term

Find all nodes that match a term_id and retrieve the first 5 nodes sorted by title. This code also puts them into a render array for display.

```

protected function loadFirstOpinion($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') //or DESC
    ->range(0, 5);
  $nids = $query->execute();
  $nodes = $storage->loadMultiple($nids);

  $render_array = [];
  foreach ($nodes as $node) {
    $render_array[] = [
      '#type' => 'markup',
      '#markup' => '<p>' . $node->getTitle(),
    ];
  }
  return $render_array;
}

```

Find 5 nodes that have a matching taxonomy term

We look for published nodes of node type `opinion` that have a term in the category field, sorted by title ascending, starting with the first result and giving us 5 results. The resulting titles are put into a render array.

```

protected function loadFirstOpinion($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') //or DESC
    ->range(0, 5);
  $nids = $query->execute();
  $nodes = $storage->loadMultiple($nids);

  $render_array = [];
  foreach ($nodes as $node) {
    $render_array[] = [
      '#type' => 'markup',
      '#markup' => '<p>' . $node->getTitle(),
    ];
  }
  return $render_array;
}

```

Find matching nodes and delete them

```

public function deleteQuery1() {
  $results = \Drupal::entityQuery('node')
    ->condition('type', 'event')
    ->range(0, 10)
    ->execute();

  if ($results) {
    foreach ($results as $result) {
      $node = Node::load($result);
      $node->delete();
    }
  }

  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => t("10 nodes deleted."),
  ];

  return $render_array;
}

```

Slice up entityQuery results into batches of 100 nodes

This is often used for batch API operations.

```
$query = \Drupal::entityQuery('node')
```

```
->condition('type', 'contract')
```

```
->condition('status', 1)
```

```
->sort('title', 'ASC');
```

```
$nids = $query->execute();
```

```
$nid_count = count($nids);
```

```
// Grab 100 nids at a time to batch process.
```

```
$batches = [];
```

```
for ($i=0;$i<=$nid_count;$i+=100) {
```

```
  $batches[] = array_slice($nids, $i, 100);
```

```
}
```

Query the creation date (among other things) using entityQuery

Note. The created (and changed) field uses a unix timestamp. This is an int 11 field in the db with a value like 1525302749 If you add a Drupal date field, its data looks like 2019-05-15T21:32:00 (varchar 20)

If you want to query a date field in a content type, you will have to fiddle around with the setTimezone stuff that is commented out below. The date field referenced below (field_date) is a standard Drupal date field.

More at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update> and

<https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

```

protected function loadOpinionForAYear($year, $term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');

  // Get a date string suitable for use with entity query.
  // $date = new DrupalDateTime(); // now
  $format = 'Y-m-d H:i';
  $start_date = DrupalDateTime::createFromFormat($format, $year . "-01-01 00:00");
  $end_date = DrupalDateTime::createFromFormat($format, $year . "-12-31 23:59");

  $start_date = $start_date->getTimestamp();
  $end_date = $end_date->getTimestamp();

  // $start_date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
  // $end_date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
  // $start_date = $start_date->format(DateTimeItemInterface::DATETIME_STORAGE_FORMAT);
  // $end_date = $end_date->format(DateTimeItemInterface::DATETIME_STORAGE_FORMAT);

  // Set the condition.
  // $query->condition('field_date.value', $start_date, '>=');
  // $query->condition('field_date.value', $end_date, '<=');

  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->condition('created', $start_date, '>=')
    ->condition('created', $end_date, '<=')
    ->sort('title', 'DESC');
  $nids = $query->execute();
  $titles = [];
  if ($nids) {
    $nodes = $storage->loadMultiple($nids);
    foreach ($nodes as $node) {
      $titles[] = $node->getTitle();
    }
  }
  return $titles;
}

```

entityQuery frequently used conditions

- Published: ->condition('status', 1)
- Text field not empty: ->condition('field_source_url', '', '<>')
- Field value > 14: ->condition('field_some_field', 14, '>')
- Reference field empty: ->notExists('field_sf_account_ref');
- Null: ->condition(\$field,NULL, '\IS NULL');
- Not Null: ->condition(\$field,NULL, '\IS NOT NULL');

Lots more at <https://www.drupal.org/docs/8/api/database-api/dynamic-queries/conditions>

Update menu items programmatically

To update several items in a menu use hook_update.

```

function park_academy_update_8002() {

$uids = \Drupal::entityQuery('menu_link_content')
->condition('menu_name', 'park-wide-utility')
->execute();

foreach($uids as $uid) {

$menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($uid);

$title = $menu_link->getTitle();
if ($title === 'Support') {
  $menu_link->set('weight', 2);
  $menu_link->set('expanded', TRUE);
// $menu_link->set('title', 'yomama');
$menu_link->set('link', 'https://www.google.com');
$menu_link->save();

}
}
}
}

```

Query multi-value fields

When querying multivalue fields, you need to use `%delta` to specify the position (or delta) for the value you are looking for. You also have to identify to the query which position (or delta) you want to query. In the example below, we specify `field_srp_voting_status.%delta` as 1 - indicating the second position (0 based always) and `field_srp_voting_status.%delta.value` for the actual value we are looking for (either accepted, rejected or incomplete):

```

$vote_number = 1;
$query = \Drupal::entityQuery('node')
->condition('type', 'correlation', '=')
->accessCheck(FALSE)
->condition('field_program', $this->programNid, '=')
->condition('field_voting_status.%delta', $vote_number, '=')
->condition('field_voting_status.%delta.value', [
  'accepted',
  'rejected',
  'incomplete'
], 'IN');

$correlation_nids = $query->execute();
$correlation_nids = array_values($correlation_nids);
return $correlation_nids;

```

Query entity reference fields if they have a value or no value

To check if there is a value in an entity reference fields, use the following code

```

$query = \Drupal::entityQuery('node')
->condition('type', 'srp_voting_record')
->accessCheck(FALSE);
if ($vote_type == 'citation') {
  // Check for empty entity reference field.
  $query->notExists('field_ref_error_feedback');
}
if ($vote_type == 'feedback_error'){
  // Check for filled entity reference field.
  $query->exists('field_ref_error_feedback');
}

```

Query entity reference fields

In the following query, we check for a value in the entity that is referenced in the entity reference field? For example, if you have an entity reference field which references node (entity) 27. This query can look in node 27 and check a field value in that node. Here we check in field_first_name for the the value Fred:

```

->condition('field_tks_pub_expectation.entity.field_first_name', 'Fred', '=')

```

For querying for a user id, we query the field_voter.entity:user.uid value. See code below:

```

protected function loadErrorFeedbackVotingRecordNode(int $user_id, int $error_feedback_nid, int $vote_number) {
  $node = [];
  $query = \Drupal::entityQuery('node')
    ->condition('type', 'srp_voting_record')
    ->condition('field_voter.entity:user.uid', $user_id)
    ->condition('field_ref_error_feedback', $error_feedback_nid)
    ->condition('field_srp_vote_number', $vote_number)
    ->accessCheck(FALSE);
  $nids = $query->execute();
  if (!empty($nids)) {
    $nid = reset($nids);
    $node = Node::load($nid);
  }
  return $node;
}

```

Static and dynamic Queries

Sometimes you will use static or dynamic queries rather than entityQueries. These use actual SQL versus the entityQuery approach where you build the various parts of the query.

Static Queries

See <https://www.drupal.org/docs/drupal-apis/database-api/static-queries>

Get a connection object

There are two ways to get a connection object:

```
/** @var \Drupal\Core\Database\Connection $connection */
$connection = Database::getConnection();

//OR

/** @var \Drupal\Core\Database\Connection $connection */
$connection = \Drupal::service('database');
```

SQL select example

Static query example from a controller. This loads some fields from the donors table and returns a render array with a count of how many results it found.

```
public function queryBuild1() {
  $database = \Drupal::database();
  $query = $database->query("SELECT id, name, amount FROM {donors}");
  $results = $query->fetchAll();
  $result_count = count($results);

  $str = "Results from db query";
  $str .= "<br/> Result count = $result_count";

  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $render_array;
}
```

Find the biggest value in a field

We make a quick query and retrieve the result. In this case we are finding the highest value for the id column.

```
public function highestId() {
  $database = \Drupal::database();
  $connection = Database::getConnection();

  $query = $connection->select('donors', 'n');
  $query->addExpression('MAX(id)', 'id');
  $result = $query->execute();
  $highest_id = intval($result->fetchField());

  $str = "Highest id = $highest_id";
  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $render_array;
}
```

SQL update query - example 1

This shows how to update a status field to the new value \$status when the uuid matches, the event is either update or add and the status is new.

```

public function setUpdateStatus(string $uuid, string $status) {
  $db_connection = \Drupal::database();
  $result = $db_connection->update('nocs_info')
    ->fields(['status' => $status])
    ->condition('uuid', $uuid)
    ->condition('event', ['UPDATE', 'ADD'], 'IN')
    ->condition('status', 'new')
    ->execute();

  return $result;
}

```

SQL update query - example 2

```

/**
 * Converts imported eventlog item/s from UPDATE to ADD by uuid.
 *
 * @param string $uuid
 *   UUID for events to convert from UPDATE to ADD.
 *
 * @return mixed
 *   Results of update query.
 */
public function convertUpdateToAddEvent(string $uuid) {
  $update_connection = \Drupal::database();
  $result = $update_connection->update('nocs_connect')
    ->fields([
      'event' => 'ADD',
    ])
    ->condition('uuid', $uuid)
    ->condition('event', 'UPDATE')
    ->condition('status', 'new')
    ->execute();

  return $result;
}

```

SQL update query - example 3

This will update values in a table and return the number of rows updated.

```

//use Drupal\Core\Database\Database;

public function updateQuery1() {
  $database = \Drupal::database();
  $query_string = "Update {donors} set amount=amount+1 where id<=10 ";
  $affectedRows = $database->query($query_string, []);
  ['return' => Database::RETURN_AFFECTED];
  $str = "Affected rows = $affectedRows";
  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
  return $render_array;
}

```

Note. This will be deprecated in Drupal 11. See

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Database%21Statement.php/function/Statement%3A%3ArowCount/9.3.x>

and

<https://git.drupalcode.org/project/drupal/-/blob/9.5.x/core/lib/Drupal/Core/Database/Connection.php#L968>

SQL insert

From <https://www.drupal.org/docs/drupal-apis/database-api/insert-queries>

Which to use? \$connection->insert() or \$connection->query() or what are the difference between insert() and query()?

- insert() has each column specified as a separate entry in the fields array and the code can clean each column value. query() has an SQL string with no way of checking individual columns.
- If you use query() with placeholders, the code can check the column values but placeholders are just an option, there is no way to ensure your SQL does not contain values not passed through placeholders.
- insert() passes the request through a set of hooks to let other modules check and modify your requests. This is the right way to work with other modules.
- query() is slightly faster because query() does not pass the request through the hooks. You might save processing time but your code will not let other modules help your code.
- insert() is more likely to work with other databases and future versions of Drupal.

SQL Insert Query

```

/**
 * @throws \Exception
 */
public function insert() {

    /** @var \Drupal\Core\Database\Connection $connection */
    $connection = \Drupal::service('database');

    // $query = $connection->insert('donors', $options);

    // single insert.
    $result = $connection->insert('donors')
        ->fields([
            'name' => 'Singleton',
            'amount' => 1,
        ])
        ->execute();

    // Note, there is an auto-increment field so insert() returns the value
    // for the new row in $result.

    $str = "Single insert returned auto-increment value of $result";

    // Multi-insert1.
    $result = $connection->insert('donors')
        ->fields(['name', 'amount'])
        ->values(['name' => 'Multiton1', 'amount' => 11])
        ->values(['name' => 'Multiton1', 'amount' => 22])
        ->execute();

    $str .= "<br/>Multi-insert1 added 2 rows";

    // Multi-insert2.
    $values = [
        ['name' => 'Multiton2', 'amount' => 111],
        ['name' => 'Multiton2', 'amount' => 222],
        ['name' => 'Multiton2', 'amount' => 333],
        ['name' => 'Multiton2', 'amount' => 444],
        ['name' => 'Multiton2', 'amount' => 555],
    ];
    $query = $connection->insert('donors')
        ->fields(['name', 'amount']);
    foreach ($values as $record) {
        $query->values($record);
    }
    $result = $query->execute();

    $str .= "<br/>Multi-insert2 added 5 rows";

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

More at <https://www.drupal.org/docs/drupal-apis/database-api/insert-queries>

SQL Delete query

This will return the number of rows affected by the SQL delete query.

```

use Drupal\Core\Database\Database;

public function deleteQuery2() {

$database = \Drupal::database();
$query_string = "Delete FROM {donors} where id>10 ";
$affectedRows = $database->query($query_string, [], ['return' => Database::RETURN_AFFECTED]);

$str = "Affected rows = $affectedRows";
$render_array['content'] = [
'#type' => 'item',
'#markup' => $str,
];
}

return $render_array;
}

```

Note. This will be deprecated in Drupal 11. See

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Database%21Statement.php/function/Statement%3A%3ArowCount/9.3.x>
also <https://git.drupalcode.org/project/drupal/-/blob/9.5.x/core/lib/Drupal/Core/Database/Connection.php#L968>.

Paragraph query

In the txg.theme file this code digs into a table for a paragraph field and grabbing the delta field value.

```

function txg_preprocess_paragraph__simple_card(&$variables) {
$card_parent = $variables['paragraph']->getParentEntity();
if ($card_parent->bundle() == 'home_aof_card') {
$variables['parent_delta'] = 0;
$database = Database::getConnection();
$result = $database->query("Select delta from node__field_para_aofs n where n.bundle = 'home_page' AND n.field_para_aofs_target_id = :target_id", [':target_id' => $card_parent->id()]);
);
if ($result) {
while ($row = $result->fetchAssoc()) {
$variables['parent_delta'] = $row['delta'];
}
}
$parent_label = $card_parent->field_ref_aof->entity->label();
$parent_path = $card_parent->field_ref_aof->entity->toUrl()->toString();
$variables['parent_label'] = $parent_label;
$variables['parent_path'] = $parent_path;
}
}

```

Create a custom table for your module

If you need a custom database table (or two) for use in a custom module, you can use hook_schema in your module.install file. This will cause the table(s) to be created at module install time and **removed** at module uninstall time.

```

function nocs_connect_schema() {
$schema['nocs_connect'] = [
'description' => 'Stores data from event log used to import/update content to site.',
'fields' => [
'id' => [
'type' => 'int',
'not null' => TRUE,
'description' => 'Primary Key: Unique ID of event log event.',
],
]
}

```

```

'uuid' => [
    'type' => 'varchar',
    'length' => 255,
    'not null' => TRUE,
    'description' => "Unique ID for content created or updated.",
],
'nid' => [
    'type' => 'varchar',
    'length' => 255,
    'not null' => FALSE,
    'description' => "Unique ID for content created or updated.",
],
'event' => [
    'type' => 'varchar',
    'length' => 255,
    'not null' => TRUE,
    'description' => 'Type of event e.g. UPDATE or ADD.',
],
'created' => [
    'type' => 'int',
    'not null' => TRUE,
    'description' => 'Content creation date.',
],
'updated' => [
    'type' => 'int',
    'not null' => TRUE,
    'description' => 'Content update date.',
],
'type' => [
    'type' => 'varchar',
    'length' => 255,
    'not null' => TRUE,
    'description' => 'Type of content created or updated.',
],
'version' => [
    'type' => 'int',
    'not null' => TRUE,
    'description' => 'Content version number.',
],
'status' => [
    'type' => 'varchar',
    'length' => 255,
    'not null' => TRUE,
    'default' => 'NEW',
    'description' => 'Status of event log row - NEW, PROCESSING, COMPLETE, or ERROR.'
],
],
'primary key' => ['id'],
'indexes' => [
    'uuid' => ['uuid'],
    'event' => ['event'],
    'type' => ['type'],
],
];

return $schema;
}

```

Reference

- [API documentation for query conditions](#)
 - [Entity query cheat sheet](#)
 - [Static queries](#)
 - [Dynamic Queries](#)
 - [Insert Queries](#)
 - [Querying date fields from 2018](#)
 - [Querying date fields](#)
-
-

[Back to top](#)

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

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

Redirects

TABLE OF CONTENTS

- [Redirect to an internal url](#)
- [Redirect in a form](#)
- [Redirect off-site \(to a third-party URL\)](#)
- [Redirect to an existing route with an anchor \(or fragment\)](#)
- [Redirect to a complex route](#)
- [Redirect in a controller](#)
- [Redirect user after login](#)
- [Redirect to the 403 or 404 page](#)
- [Redirect to a new page after node operation](#)
- [Redirect dynamically to wherever you came from](#)

views 72

Redirect to an internal url

In a controller when we return a `RedirectResponse` instead of a render array, Symfony redirects to the URL specified in the `RedirectResponse`.

For example:

```
use \Symfony\Component\HttpFoundation\RedirectResponse;

return new RedirectResponse('node/1');

// Or to redirect to the front page of the site.

$url = Url::fromRoute('<front>');
return new RedirectResponse($url->toString());
```

Redirect in a form

In a form, you can redirect to a route by its name:

```
$form_state->setRedirect('entity.bike_part.canonical', ['bike_part' => $entity->id()]);
```

Note. You can also redirect to a specific id (anchor) on the page by adding the fragment parameter

```
$form_state->setRedirect('tea_teks_admin.timeline_detail',
['node'=>$parent_nid],
['fragment' => 'milestone-' . $nid,]
);
```

Or here in `submitForm()`:

```
$url = Url::fromRoute('user_account.user_register', [], ['query' => ['destination' => $shippingUrl]]);
$form_state->setRedirectUrl($url);
```

Redirect off-site (to a third-party URL)

From <https://drupal.stackexchange.com/questions/136641/how-do-i-redirect-to-an-external-url>

Redirect to a third party website with Redirect permanent status. The URL must have absolute path like <http://www.google.com>

```
return new RedirectResponse('https://google.com');
```

e.g.

```
Use Drupal\Core\Routing\TrustedRedirectResponse;
```

```
$absolute_url = 'https://google.com';
$response = new RedirectResponse($absolute_url, 301);
//OR
$response = new \Drupal\Core\Routing\TrustedRedirectResponse($absolute_url, 301);
$response->send();

exit(0);
```

Redirect to an existing route with an anchor (or fragment)

You can specify the route using the route name from your module.routing.yml file.

Note. You can also redirect to a specific id (anchor) on the page by adding the fragment parameter

```
$form_state->setRedirect('tra_teks_admin.timeline_detail',
['node' => $parent_nid],
['fragment' => 'milestone-' . $nid,]
);
```

Redirect to a complex route

Here the route requires four arguments so they all need to be passed in.

```
$form_state->setRedirectUrl(Url::fromRoute('tra_teks_srp.confidential_voting', [
'citadel_nid' => $citadel_nid,
'performance_nid' => $performance_nid,
'action' => 'vote',
'type' => $type,
]));
```

Redirect in a controller

When you need to send the user to a different page on your site based on some logic, you might use code something like this. Lots of processing happens and if the conditions are met, instead of returning a render array, we return a RedirectResponse and the browser will load that page.

```

protected function reloadOperations(): ?RedirectResponse {
    $program_nid = $this->programNode->id();
    if (substr($this->action, 0, 6) === 'reload') {
        $new_type = substr($action, 7);
        $this->votingProcessor->buildVotingPath($this->programNode, $new_type);
        $valid_voting_path = $this->votingProcessor->loadYourVotingPathFromTempStore($program_nid);
        $empty_voting_path = $this->votingProcessor->isEmptyVotingPath();
        if ($valid_voting_path && $empty_voting_path) {
            // Redirect to kss overview.
            $message = "No eligible items found. Redirected to KSS overview";
            \Drupal::messenger()->addMessage($message);
            $url = Url::fromRoute('tea_teks_srp.kss_overview', [
                'program' => $program_nid,
                'type' => 'all',
                'userid' => $this->persona->getUserId(),
            ]);
            return new RedirectResponse($url->toString());
        }
        if ($valid_voting_path && !$empty_voting_path) {
            //redirect to the first item in the path.
            $item = $this->votingProcessor->getFirstVotingPathItem();
            $vote_type = 'narrative';
            if (empty($item['narrative'])) {
                $vote_type = 'activity';
            }
            $url = Url::fromRoute('tea_teks_srp.correlation_voting', [
                'program' => $program_nid,
                'expectation' => $item['expectation_nid'],
                'correlation' => $item['correlation_nid'],
                'action' => "vote-$new_type",
                'type' => $vote_type,
            ]);
            return new RedirectResponse($url->toString());
        }
    }
}

```

Redirect user after login

From: <https://www.drupal.org/forum/support/module-development-and-code-questions/2013-08-18/how-to-redirect-user-after-login-in> This example checks both the route and user role to conditionally redirect.

```

use Drupal\Core\Session\AccountInterface;
use Drupal\Core\Url;

/**
 * Implements hook_user_login().
 */
function greenacorn_user_login(AccountInterface $account) {
    $roles = $account->getRoles();
    $route_name = \Drupal::routeMatch()->getRouteName();
    if ($route_name != 'user.reset.login' && in_array('client', $roles)) {
        $destination = Url::fromUserInput('/my-issues')->toString();
        \Drupal::service('request_stack')->getCurrentRequest()->query->set('destination', $destination);
    }
}

```

Redirect to the 403 or 404 page

```
// Redirect to the 403 page.
throw new \Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException();

// Redirect to the 404 page.
throw new \Symfony\Component\HttpKernel\Exception\NotFoundHttpException();
```

Redirect to a new page after node operation

This was implemented in a module in a hook_form_alter call. Here we tell the form to call the special handler: cn_submit_handler() function in the form.

```
function obg_mods_form_alter(array &$form, FormStateInterface $form_state, $form_id) {
  $accountProxy = \Drupal::currentUser();
  $account = $accountProxy->getAccount();

  // Add special validation for anonymous users only.
  if (($accountProxy->isAnonymous() && ($form_id == 'node_catastrophe_notice_form'))) {
    $form['#validate'][] = 'cn_form_validate';

    // Submit handler to redirect to /thanks-your-submission.
    foreach (array_keys($form['actions']) as $action) {
      if ($action != 'preview' && !isset($form['actions'][$action]['#type']) && $form['actions'][$action]['#type'] === 'submit') {
        $form['actions'][$action]['#submit'][] = 'cn_submit_handler';
      }
    }
  }
}
```

And here is the submit handler where the work gets done and you are redirected to /thanks-your-submission.

```
/*
 * Submit handler to redirect user to thank-you page.
 *
 * @param $form
 * @param \Drupal\Core\Form\FormStateInterface $form_state
 */
function cn_submit_handler( $form, FormStateInterface $form_state ) {
  $url = Url::fromUri('internal:/thanks-your-submission');
  $form_state->setRedirectUrl($url);
}
```

More at <https://drupal.stackexchange.com/questions/163626/how-to-perform-a-redirect-to-custom-page-after-node-save-or-delete>

Redirect dynamically to wherever you came from

In this case, in a form, we grab the referrer url in the buildForm method using the following code:

```
public function buildForm(array $form, FormStateInterface $form_state, int $program_nid = 0, int $feedback_error_nid = 0, int $citation_nid = 0){  
    $form['#theme'] = 'tea_teks_srp__vote_error';  
  
    // Get the referer.  
    $request = \Drupal::request();  
    $referer = $request->headers->get('referer');  
    // $base_url = Request::createFromGlobals()->getSchemeAndHttpHost();  
    $base_url = \Drupal::request()->getSchemeAndHttpHost();  
    $alias = '';  
    if (!is_null($referer)) {  
        $alias = substr($referer, strlen($base_url));  
    }  
    $form_state->set('referrer_alias', $alias);  
    ...  
}
```

Then in the submitForm() method, once we've completed the work we needed to do, we can redirect back to where we came from using the code below. We can optionally add a fragment which refers to an ID on the page.

```
$referrer_alias = $form_state->get('referrer_alias');  
  
// Add the fragment so they drop back on the item they came from.  
$url = Url::fromUri('internal' . $referrer_alias, ['fragment' => "item_$feedback_error_nid"]);  
$form_state->setRedirectUrl($url);
```

[Back to top](#)

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

Page last modified: Jun 22 2023.

[Edit this page on GitHub](#)

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

Render Arrays

TABLE OF CONTENTS

- [Overview](#)
- [Overview of the Theme system and Render API.](#)
- [Caching](#)
- [Properties](#)
- [Image](#)
- [Simple Text](#)
- [Text with variable substitution \(Placeholders\)](#)
- [Wrap an element with a div with a class](#)
- [Prefix and suffix](#)
- [Date](#)
- [Image](#)
- [Several Url's.](#)
- [Two paragraphs](#)
- [A button that opens a modal dialog](#)
- [A link](#)
- [A link with a class](#)
- [A link and its TWIG template](#)
- [A link with parameters and a template file](#)
- [Simple unordered list](#)
- [Unordered list of links for a menu](#)
- [Nested Unordered List](#)
- [Select \(dropdown\)](#)
- [Select \(dropdown\) Ajaxified](#)
- [Limit allowed tags in markup](#)
- [Disable an element](#)
- [Resources](#)

views | 141

Overview

Render Arrays are the building blocks of a Drupal page. A render array is an associative array which conforms to the standards and data structures used in Drupal's Render API. The Render API is also integrated with the Theme API.

In many cases, the data used to build a page (and all parts of it) is kept as structured arrays until the final stage of generating a response. This provides enormous flexibility in extending, slightly altering or completely overriding parts of the page.

Render arrays are nested and thus form a tree. Consider them Drupal's "render tree" — Drupal's equivalent of the DOM.

Note: While render arrays and arrays used by the Form API share elements, properties and structure, many properties on form elements only have meaning for the Form API, not for the Render API. Form API arrays are transformed into render arrays by FormBuilder. Passing an unprocessed Form API array to the Render API may yield unexpected results.

Here is a simple render array that displays some text.

```
$my_render_array['some_item'] = [  
  '#type' => markup,  
  '#markup' => "This is a test",  
];
```

All forms are Render arrays. This is important when need to use the form API to create forms.

This is mostly from <https://www.drupal.org/docs/drupal-apis/render-api/render-arrays>

Overview of the Theme system and Render API.

The main purpose of Drupal's Theme system is to give themes complete control over the appearance of the site, which includes the markup returned from HTTP requests and the CSS files used to style that markup. In order to ensure that a theme can completely customize the markup, module developers should avoid directly writing HTML markup for pages, blocks, and other user-visible output in their modules, and instead return structured "render arrays". Doing this also increases usability, by ensuring that the markup used for similar functionality on different areas of the site is the same, which gives users fewer user interface patterns to learn.

From the Render API overview at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/theme_render/10.0.x

Caching

You can specify caching information when creating render arrays. Cache keys, cache contexts, cache tags and cache max-age can all be defined.

The Drupal rendering process has the ability to cache rendered output at any level in a render array hierarchy. This allows expensive calculations to be done infrequently, and speeds up page loading. See the [Cache API topic](#) for general information about the cache system.

In order to make caching possible, the following information needs to be present:

- **Cache keys:** Identifiers for cacheable portions of render arrays. These should be created and added for portions of a render array that involve expensive calculations in the rendering process.
- **Cache contexts:** Contexts that may affect rendering, such as user role and language. When no context is specified, it means that the render array does not vary by any context.
- **Cache tags:** Tags for data that rendering depends on, such as for individual nodes or user accounts, so that when these change the cache can be automatically invalidated. If the data consists of entities, you can use [\Drupal\Core\Entity\EntityInterface::getCacheTags\(\)](#) to generate appropriate tags; configuration objects have a similar method.
- **Cache max-age:** The maximum duration for which a render array maybe cached. Defaults to [\Drupal\Core\Cache\Cache::PERMANENT](#) (permanently cacheable).

Cache information is provided in the #cache property in a render array. In this property, always supply the cache contexts, tags, and max-age if a render array varies by context, depends on some modifiable data, or depends on information that's only valid for a limited time, respectively. Cache keys should only be set on the portions of a render array that should be cached. Contexts are automatically replaced with the value for the current request (e.g. the current language) and combined with the keys to form a cache ID. The cache contexts, tags, and max-age will be propagated up the render array hierarchy to determine cacheability for containing render array sections.

Here's an example of what a #cache property might contain:

```
'#cache' => [
  'keys' => ['entity_view', 'node', $node->id()],
  'contexts' => ['languages'],
  'tags' => $node->getCacheTags(),
  'max-age' => Cache::PERMANENT,
],
```

At the response level, you'll see X-Drupal-Cache-Contexts and X-Drupal-Cache-Tags headers.

Reproduced from the Render API overview at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/theme_render/10.0.x

Properties

Elements that start with # are properties and can include the following: #type, #theme, #markup, #prefix, #suffix, #plain_text OR #allowed_tags.

Render arrays (at any level of the hierarchy) will usually have one of the following properties defined:

- **#type:** Specifies that the array contains data and options for a particular type of "render element" (for example, 'form', for an HTML form;

'textfield', 'submit', for HTML form element types; 'table', for a table with rows, columns, and headers). See [Render elements](#) below for more on render element types.

- **#theme**: Specifies that the array contains data to be themed by a particular theme hook. Modules define theme hooks by implementing `hook_theme()`, which specifies the input "variables" used to provide data and options; if a `hook_theme()` implementation specifies variable 'foo', then in a render array, you would provide this data using property '#foo'. Modules implementing `hook_theme()` also need to provide a default implementation for each of their theme hooks, normally in a Twig file. For more information and to discover available theme hooks, see the documentation of `hook_theme()` and the [Default theme implementations topic](#).
- **#markup**: Specifies that the array provides HTML markup directly. Unless the markup is very simple, such as an explanation in a paragraph tag, it is normally preferable to use `#theme` or `#type` instead, so that the theme can customize the markup. Note that the value is passed through [\Drupal\Component\Utility\Xss::filterAdmin\(\)](#), which strips known XSS vectors while allowing a permissive list of HTML tags that are not XSS vectors. (For example, `<script>` and `<style>` are not allowed.) See [\Drupal\Component\Utility\Xss::\\$adminTags](#) for the list of allowed tags. If your markup needs any of the tags not in this list, then you can implement a theme hook and/or an asset library. Alternatively, you can use the key `#allowed_tags` to alter which tags are filtered.
- **#plain_text**: Specifies that the array provides text that needs to be escaped. This value takes precedence over `#markup`.
- **#allowed_tags**: If `#markup` is supplied, this can be used to change which tags are allowed in the markup. The value is an array of tags that `Xss::filter()` would accept. If `#plain_text` is set, this value is ignored.

Usage example:

```
$output['admin_filtered_string'] = [
  '#markup' => '<em>This is filtered using the admin tag list</em>',
];
$output['filtered_string'] = [
  '#markup' => '<video><source src="v.webm" type="video/webm"></video>',
  '#allowed_tags' => [
    'video',
    'source',
  ],
];
$output['escaped_string'] = [
  '#plain_text' => '<em>This is escaped</em>',
];

```

JavaScript and CSS assets are specified in the render array using the `#attached` property (see [Attaching libraries in render arrays](#)).

From https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/theme_render/10.0.x

And

```
$variables['content']['field_image'][">#suffix'] = "this is a suffix to the image";
```

Or

```
$variables['content']['custom_field']= ['#type'=>'markup', '#markup'=>'Hello World'];
```

You can make up your own properties e.g. see the `#selwyn_id` below for use elsewhere.

```

$citation_nid = 25;
$form['actions']['accept'] = [
  '#type' => 'submit',
  '#value' => $this->t("Accept Citation $citation_nid"),
  '#citation_nid' => $citation_nid,
  '#voting_action' => 'Accept',
  '#name' => "accept_citation_$citation_nid",
  '#selwyn_id' => ['edit-accept-' . $citation_nid],
  '#attributes' => [
    'class' => [
      'hilited-button',
      'blue-button',
    ],
    'id' => ['edit-accept-' . $citation_nid],
  ],
];

```

Image

```

$image = [
  '#theme'=>'image',
  '#uri' => 'public://photo.jpg',
  '#alt' => 'hello'
];

```

Simple Text

```

$text_array = [
  '#markup' => $this->t('Hello world!'),
];

```

Text with variable substitution (Placeholders)

```

$render_array = [
  '#type' => 'markup',
  '#markup' => $this->t('You are viewing @title. Unfortunately there is no image defined for delta: @delta.', ['@title' => $node->getTitle(), '@delta' =>$delta]),
];

```

And from the Render API Overview at

https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/group/theme_render/10.0.x :

Placeholders in render arrays

Render arrays have a placeholder mechanism, which can be used to add data into the render array late in the rendering process. This works in a similar manner to [\Drupal\Component\Render\FormattableMarkup::placeholderFormat\(\)](#), with the text that ends up in the #markup property of the element at the end of the rendering process getting substitutions from placeholders that are stored in the 'placeholders' element of the #attached property.

For example, after the rest of the rendering process was done, if your render array contained:

```

$build['my_element'] = [
  '#markup' => 'Something about @foo',
  '#attached' => [
    'placeholders' => [
      '@foo' => ['#markup' => 'replacement'],
    ],
];

```

then #markup would end up containing 'Something about replacement'.

Note that each placeholder value *must* itself be a render array. It will be rendered, and any cache tags generated during rendering will be added to the cache tags for the markup.

Wrap an element with a div with a class

```
$ra['list'][$customer]['name'] = [
  '#prefix' => '<div class="customer-name">',
  '#suffix' => '</div>',
  '#type' => 'markup',
  '#markup' => $customer['name'],
];
```

Prefix and suffix

```
$ra['#prefix'] = '<div id="option-landing-block">';
$ra['#suffix'] = '</div>';
```

Date

To create a date object and return the year in a render array, use this code. Here is the code from a block build method:

```
public function build() {
  $date = new \DateTime();
  return [
    '#markup' => t('Copyright @year© My Company', [
      '@year' => $date->format('Y'),
    ]),
  ];
}
```

This uses Drupal\Core\Datetime\DrupalDateTime which is just a wrapper for \DateTime.

Image

Load an image and display it with the alt text

```
public function displayProductImage(NodeInterface $node, $delta) {
  if (isset($node->field_product_image[$delta])) {
    $imageData = $node->field_product_image[$delta]->getValue();
    $file = File::load($imageData['target_id']);
    $render_array['image_data'] = array(
      '#theme' => 'image_style',
      '#uri' => $file->getFileUri(),
      '#style_name' => 'product_large',
      '#alt' => $imageData['alt'],
    );
  }
}
```

Several Url's.

This queries for some nodes, generate a list of url's and returns them as a render array. The '#list_type' => 'ol' (or ordered list)

```

use Drupal\Core\Url;

public function build(){
  $result = $this->nodeStorage->getQuery()
    ->accessCheck(TRUE)
    ->condition('type', 'water_action')
    ->condition('status', 1)
    ->range(0, $this->configuration['block_count'])
    ->sort('title', 'ASC')
    ->execute();

  if ($result) {
    //Only display block if there are items to show.
    $items = $this->nodeStorage->loadMultiple($result);

    $build['list'] = [
      '#theme' => 'item_list',
      '#items' => [],
    ];
    foreach ($items as $item) {
      $translatedItem = $this->entityRepository->getTranslationFromContext($item);
      $nid = $item->id();
      $url = Url::fromUri("internal:/node/$nid");

      $build['list'][">#items'][$item->id()] = [
        '#title' => $translatedItem->label(),
        '#type' => 'link',
        '#url' => $url,
      ];
    }
  }
}

```

Two paragraphs

```

$rArray = [
  'first_para' => [
    '#type' => 'markup',
    '#markup' => '...para 1 here....<br>',
  ],
  'second_para' => [
    '#type' => 'markup',
    '#markup' => '...para 2 here....<br>',
  ],
];
return $rArray;

```

A button that opens a modal dialog

```

use Drupal\Core\Url;

public function build() {
  $link_url = Url::fromRoute('custom_modal.modal');
  $link_url->setOptions([
    'attributes' => [
      'class' => ['use-ajax', 'button', 'button--small'],
      'data-dialog-type' => 'modal',
      'data-dialog-options' => Json::encode(['width' => 400]),
    ]
  ]);
}

return [
  '#type' => 'markup',
  '#markup' => Link::fromTextAndUrl(t('Open modal'), $link_url)->toString(),
  '#attached' => ['library' => ['core/drupal.dialog.ajax']]
];

```

A link

Here is a simple link

```
use Drupal\Core\Url;
```

```

$form['noaccount'] = [
  '#type' => 'link',
  '#title' => $this->t('Continue without account'),
  '#url' => Url::fromRoute('front'),
];

```

Other possible urls:

```

'#url' => Url::fromUri('internal:/dashboard'),
'#url' => Url::fromUri('internal:/node/360'),
'#url' => Url::fromUri('mailto:' . $value),

```

A link with a class

Here we add the #attributes to wrap the link in the classes: button, button-action, button--primary, and button--small:

```

$form['button'] = [
  '#type' => 'link',
  '#url' => Url::fromUri('internal:/dashboard'),
  '#title' => $this->t('Go to My Training'),
  '#attributes' => ['class' => ['button', 'button-action', 'button--primary', 'button--small']],
];

```

A link and its TWIG template

Here is a link with the details of what you expect to see in the TWIG template:

```

use Drupal\Core\Url;

$back_home_link = [
  '#type' => 'link',
  '#title' => $this->t('Continue without account'),
  '#url' => Url::fromRoute('<front>'),
];

```

\$variables['back_home_link'] = \$back_home_link

and in the template you would expect to see something like:

```

{{ content.back_home_link }}

```

A link with parameters and a template file

This path takes a 4 parameters. Here is its path as defined in the `routing.yml` file:

```

team_abc.correctional_voting:
path: '/team/abc/admin//program/{program}/expectation/{expectation}/correlation/{correlation}/{action}/{type}'
defaults:
  _controller: '\Drupal\team_abc\Controller\CorrelationVotingController::content'
  _title: 'Correctional Voting'
requirements:
  _permission: 'manage voting process'
options:
parameters:
  program:
    type: entity:node
  expectation:
    type: entity:node
  correlation:
    type: entity:node
  no_cache: TRUE'

```

Note the options in the `routing.yml` file which automatically convert the node ids to actual entities (Drupal loads the nodes internally) and passes those to the controller.

Then in the controller, we build a URL, specifying the parameters:

```

$url = Url::fromRoute('team_abc.correctional_voting', [
  'program' => $program->id(),
  'expectation' => $next_breakout_path_item['expectation_nid'],
  'correlation' => $next_breakout_path_item['correlation_nid'],
  'action' => 'vote',
  'type' => 'narrative'
]);

```

\$next_breakout = [

- '#type' => 'link',
- '#title' => t('Next Breakout'),
- '#url' => \$url,

];

// ...

\$next_links[] = \$next_breakout;

Then we wrap up all the variables and send them to `buildDetails`

```

$content = [
// ...
'program' => $program_info,
'previous_links' => $previous_links,
'next_links' => $next_links,
'expectation_citem_text' => strip_tags($expectation_citem_text),
];
return $this->buildDetails($content, $breadcrumbs, $management_links, $correlation_info, $citations);
}

```

Which wraps the content in an array for rendering in twig. Note below that the `#theme` property which identifies the template filename. The `#theme: team_abc_correctional_voting` translates to the twig template file: `team-abc--correctional-voting.html.twig` where the underscores become dashes.

```

public function buildDetails(array $content, array $breadcrumbs, array $management_links, array $correlation_info, array $citations): array {
    return [
        '#theme' => 'team_abc__correctional_voting',
        '#content' => $content,
        '#breadcrumbs' => $breadcrumbs,
        '#management_links' => $management_links,
        '#correlation' => $correlation_info,
        '#citations' => $citations,
    ];
}

```

Then in `team-abc--correctional-voting.html.twig` the `next` links are rendered – see [this example](#)

```

<div class="cell small-12 medium-6">
{%
    if content.next_links %}
    <ul class="no-bullet nav-links prev">
        {% for next_link in content.next_links %}
            Move mouse below for Next invisible links<li>{{ next_link }}</li>
        {% endfor %}
    </ul>
{%
    endif %}
</div>

```

This example may be a little confusing as it loops through an array of links.

Simple unordered list

From: <https://drupal.stackexchange.com/questions/214928/create-unordered-list-in-render-array>

```

$content = [
    '#theme' => 'item_list',
    '#list_type' => 'ul',
    '#title' => 'My List',
    '#items' => ['item 1', 'item 2'],
    '#attributes' => ['class' => 'mylist'],
    '#wrapper_attributes' => ['class' => 'container'],
];

```

Unordered list of links for a menu

Here a list of links is created in a controller:

```

$content['tabs'] = [
    '#theme' => 'item_list',
    '#list_type' => 'ul',
    '#items' => [
        [
            '#type' => 'link',
            '#title' => $this->t('My Plumbing Training'),
            '#url' => Url::fromRoute('abc_academy.dashboard_tab', ['tab' => 'online'])
        ],
        [
            '#type' => 'link',
            '#title' => $this->t('My Enrollment Training'),
            '#url' => Url::fromRoute('abc_academy.dashboard_tab', ['tab' => 'enrollment'])
        ],
        [
            '#type' => 'link',
            '#title' => $this->t('My Transcript'),
            '#url' => Url::fromRoute('abc_academy.dashboard_tab', ['tab' => 'transcript'])
        ],
        [
            '#type' => 'link',
            '#title' => $this->t('My Certificates'),
            '#url' => Url::fromRoute('abc_academy.dashboard_tab', ['tab' => 'transcript'])
        ],
        [
            '#type' => 'link',
            '#url' => Url::fromUri('internal:/dashboard/certificates')
        ]
    ];
];

return $content;

```

Nested Unordered List

```

$sidebar = [
    '#title' => 'My List',
    '#theme' => 'item_list',
    '#list_type' => 'ul',
    '#attributes' => ['class' => 'mylist'],
    '#wrapper_attributes' => ['class' => 'container'],
    '#items' => [
        [
            '#type' => 'link',
            '#title' => t('My Online Training'),
            '#url' => Url::fromUri('internal:/node/1')
        ],
        [
            '#type' => 'link',
            '#title' => t('My Instructor-led Training'),
            '#url' => Url::fromUri('internal:/node/2')
        ],
        ['#markup' => '<ul><li>item1</li><li>item2</li></ul>'],
        ['#markup' => '<ul><li>item1</li><li>item2</li><li>item3</li></ul>'],
    ],
];

```

Select (dropdown)

To build a select element, fill an array with some keys and text labels. The text labels will appear in the dropdown.

To set the default value i.e. the value that appears in the dropdown when it is first displayed, specify the key. For example: If the contents of

the dropdown are an array like ['aaa', 'vvv', 'zzz'] then you can specify \$default=2 to display zzz as the default.

In the example below, the default is set to /node/364 and the dropdown will display Above ground pool 1, Above ground pool 2 etc.

```
// Select element.  
$options = [  
  '/node/360' =>'Above ground pool 1',  
  '/node/362' =>'Above ground pool 2',  
  '/node/364' =>'Underground pool',  
  '/node/359' =>'Patio pool',  
];  
  
// Set the default value - it must be the key.  
$default = '/node/364';  
$form['select'] = [  
  '#type' => 'select',  
  '#title' => $this->t('Select video'),  
  '#description' => 'Test Description',  
  '#default_value' => $default,  
  '#options' => $options,  
];
```

Select (dropdown) Ajaxified

Often sites need select elements that do some action e.g. redirect when the user makes a selection in the dropdown. Here is one example such a select element. I populate the \$options with the result of a database query. When the user changes the selection in the dropdown, it calls the callback videoSelectChange(). The callback redirects to the URL in question using the \$command = new RedirectComand();

```
$form['select'] = [  
  '#type' => 'select',  
  //  '#title' => $this->t('Select video'),  
  '#description' => 'Test Description',  
  '#default_value' => $default,  
  '#options' => $options,  
  '#ajax' => [  
    'callback' => [$this, 'videoSelectChange'],  
    'event' => 'change',  
    'wrapper' => $ajax_wrapper,  
  ],  
];  
  
/**  
 * Callback function for changes to the select elements.  
 */  
  
public function videoSelectChange(array $form, FormStateInterface $form_state) {  
  $values = $form_state->getValues();  
  $elem = $form_state->getTriggeringElement();  
  $response = new AjaxResponse();  
  $url = Url::fromUri('internal:' . $values[$elem["#name"]]);  
  $command = new RedirectCommand($url->toString());  
  $response->addCommand($command);  
  return $response;  
}
```

Limit allowed tags in markup

Here we allow <i> (italics) tags in the login menu item, and <i> and <sup> (superscript) tags in the logout menu item

```

/**
 * Implements hook preprocess_menu().
 */
function postal_theme preprocess_menu(&$vars, $hook) {
  if ($hook == 'menu__account') {
    $items = $vars['items'];
    foreach ($items as $key => $item) {
      if ($key == 'user.page') {
        $vars['items'][$key]['title'] = [
          '#markup' => 'Log the <i>flock</i><sup>TM</sup> in',
          '#allowed_tags' => ['i'],
        ];
      }
      if ($key == 'user.logout') {
        $vars['items'][$key]['title'] = [
          '#markup' => 'Log the <i>flock</i> <sup>TM</sup>out',
          '#allowed_tags' => ['i', 'sup'],
        ];
      }
    }
  }
}

```

Disable an element

In this example the `accept` button is disabled when `$my_current_vote` is accepted.

```

$form['accept'] = [
  '#type' => 'submit',
  '#value' => $value,
  '#name' => "accept_$feedback_error_nid",
  '#voting_action' => 'accepted',
  '#prefix' => '<div class="srp-vote-form">',
  '#id' => 'edit-accept' . $feedback_error_nid,
  '#attributes' => [
    'class' => [
      'hilited-button',
      'blue-button',
      'accept-button',
    ],
  ],
];
}

if (strtolower($my_current_vote) == 'accepted') {
  $form['accept']['#attributes']['class'][] = 'selected';
  $form['reject']['#disabled'] = TRUE;
}

```

Resources

- [Render API overview for Drupal 10](#)
- [Render Arrays from Drupal.org updated August 2022](#)

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

Routes and Controllers

TABLE OF CONTENTS

- [Overview](#)
 - [Route](#)
 - [Controller](#)
 - [Connecting to a twig template](#)
- [Simple page without arguments](#)
- [Page with arguments](#)
- [Simple form](#)
- [Admin form \(or settings form\)](#)
- [Routing permissions](#)
 - [A specific permission](#)
 - [Multiple permissions](#)
- [Set the page title dynamically](#)
- [Disable caching on a route](#)
- [Generate route and controller with Drush](#)
- [Finding routes with Drush](#)
 - [All routes](#)
 - [Specific path](#)
 - [Specific route name](#)
- [Getting some help from Chat GPT](#)
- [Resources](#)

views 105

Overview

Route

A route connects a URL path to a controller. In `hello_world.routing.yml` (e.g. in `modules/custom/hello_world/hello_world.routing.yml`) The path `/hello` maps to the controller `HelloWorldController` and the member function: `helloWorld()`. When a user visits `/hello`, Drupal checks to see that the user has `access content` permission and the `helloWorld()` function is executed.

```
hello_world.hello:  
  path: '/hello'  
  defaults:  
    _controller: '\Drupal\hello_world\Controller\HelloWorldController::helloWorld'  
    _title: 'Our first route'  
  requirements:  
    _permission: 'access content'
```

Controller

This is the PHP function that takes info from the HTTP request and constructs and returns an HTTP response (as a Symfony `ResponseObject`). The controller contains your logic to render the content of the page.

The controller will usually return a render array but they can return an HTML page, an XML document, a serialized JSON array, an image, a redirect, a 404 error or almost anything else.

A simple render array looks like this:

```
return [
  '#markup' => 'blah',
]
```

Connecting to a twig template

Most often, you will have a twig template connected to your controller. You do this by a combination of a `#theme` element in the render array and a `hook_theme` function in a `.module` file.

In the example below, the controller returns a large render array and the theme is identified as `abc_teks_srp__correlation_voting`.

```
return [
  '#theme' => 'abc_teks_srp__correlation_voting',
  '#content' => $content,
  '#breadcrumbs' => $breadcrumbs,
  '#management_links' => $management_links,
  '#correlation' => $correlation_info,
  '#citations' => $citations,
];
```

In a module file, there is a `hook_theme` function which corresponds to the `abc_teks_srp_theme` and identifies the template name as `abc-teks-srp-correlation-voting`. Here is the significant part of the `hook_theme()` function

```
/**
 * Implements hook_theme().
 */
function abc_teks_srp_theme() {
  $variables = [
    'abc_teks_srp' => [
      'render_element' => 'children',
    ],
    'abc_teks_srp__correlation_voting' => [
      'variables' => [
        'content' => NULL,
        'breadcrumbs' => NULL,
        'management_links' => NULL,
        'correlation' => NULL,
        'citations' => NULL,
      ],
      'template' => 'abc-teks-srp--correlation-voting',
    ],
  ];
}
```

The template will therefore be `abc-teks-srp--correlation.voting.yml`

Simple page without arguments

This route is for a page with no arguments/parameters.

In the file `page_example.routing.yml` (e.g. `web/modules/contrib/examples/page_example/page_example.routing.yml` and the controller is at `web/modules/contrib/examples/page_example/src/Controller/PageExampleController.php`

```

# If the user accesses https://example.com/?q=examples/page-example/simple,
# or https://example.com/examples/page-example/simple,
# the routing system will look for a route with that path.
# In this case it will find a match, and execute the _controller callback.
# Access to this path requires "access simple page" permission.

page_example_simple:
  path: 'examples/page-example/simple'
  defaults:
    _controller: '\Drupal\page_example\Controller\PageExampleController::simple'
    _title: 'Simple - no arguments'
  requirements:
    _permission: 'access simple page'

```

Page with arguments

From web/modules/contrib/examples/page_example/page_example.routing.yml {first}/{second} are the arguments.

```

# Since the parameters are passed to the function after the match, the
# function can do additional checking or make use of them before executing
# the callback function. The placeholder names "first" and "second" are
# arbitrary but must match the variable names in the callback method, e.g.
# "$first" and "$second".

page_example_arguments:
  path: 'examples/page-example/arguments/{first}/{second}'
  defaults:
    _controller: '\Drupal\page_example\Controller\PageExampleController::arguments'
  requirements:
    _permission: 'access arguments page'

```

Simple form

From web/modules/custom/rsvp/rsvp.routing.yml. This route will cause Drupal to load the form: RSVPForm.php so the user can fill it out.

```

rsvp.form:
  path: '/rsvplist'
  defaults:
    _form: 'Drupal\rsvp\Form\RSVPForm'
    _title: 'RSVP to this Event'
  requirements:
    _permission: 'view rsvplist'

```

Admin form (or settings form)

From web/modules/custom/rsvp/rsvp.routing.yml this route loads the admin or settings form RSVPConfigurationForm.

```

rsvp.admin_settings:
  path: '/admin/config/content/rsvp'
  defaults:
    _form: 'Drupal\rsvp\Form\RSVPConfigurationForm'
    _title: 'RSVP Configuration Settings'
  requirements:
    _permission: 'administer rsvplist'
  options:
    _admin_route: TRUE

```

Routing permissions

These are defined in your module.permissions.yml e.g. rsvp.permissions.yml. If you add this file to a module, a cache clear will cause the new permissions to appear on the permissions page.

This requires the user to be logged in to access this route:

```
requirements:  
  _user_is_logged_in: 'TRUE'
```

To skip permissions, set `_access` to TRUE like this:

```
requirements:  
  _access: 'TRUE'
```

A specific permission

To specify a particular permission, use the following. Note. Case is critical!

```
requirements:  
  _permission: 'administer rsvplist'
```

Multiple permissions

Drupal allows stacking permissions with the plus(+) sign. Note the + sign means OR. e.g.

```
requirements:  
  _permission: 'vote on own squishy item+manage squishy process'
```

Set the page title dynamically

This code shows how to set the title statically in the `module.routing.yml` file, as well as how to call a function like `getTitle()` to return it so it can be dynamically generated:

```
org_onions_summary:  
  path: 'onions/{term_id}'  
  defaults:  
    _controller: '\Drupal\org_onions\Controller\OnionsController::buildOnionsSummary'  
    # Static Title  
    # _title: 'Opinions Summary'  
    # Dynamic Title  
    _title_callback: '\Drupal\org_onions\Controller\OnionsController::getTitle'  
  requirements:  
    _permission: 'access content'
```

In your controller, add the function `getTitle()`. This function can actually be called whatever you like.

```
/**  
 * Returns a page title.  
 */  
public function getTitle() {  
  $current_path = \Drupal::service('path.current')->getPath();  
  $path_args = explode('/', $current_path);  
  $boss_name = $path_args[2];  
  $boss_name = ucwords(str_replace('-', ' ', $boss_name));  
  
  $config = \Drupal::config('system.site');  
  $site_name = $config->get('name');  
  return $boss_name . ' Onions | ' . $site_name;  
  
  //or  
  return $boss_name . ' onions | ' . \Drupal::config('system.site')->get('name');  
}
```

Disable caching on a route

This will cause Drupal to rebuild the page internally on each page load but won't stop browsers or CDN's from caching. The line: `no_cache: TRUE` is all you need to disable caching for this route.

```
requirements:  
  _permission: 'access content'  
options:  
  no_cache: TRUE
```

Generate route and controller with Drush

Drush has the ability to generate code to start you off. Use `drush generate module` and/or `drush generate controller` to get a nice starting point for you to write your own controllers.

For more, on generating controllers see <https://www.drush.org/latest/generators/controller/>

This is what it looks like to generate a controller:

```
$ drush generate controller  
  
Welcome to controller generator!  
-----  
  
Module machine name [web]:  
► general  
  
Class [GeneralController]:  
► ExampleController  
  
Would you like to inject dependencies? [No]:  
►  
  
Would you like to create a route for this controller? [Yes]:  
►  
  
Route name [general.example]:  
► general.book_example  
  
Route path [/general/example]:  
► /general/book_example  
  
Route title [Example]:  
► Book Example  
  
Route permission [access content]:  
►  
  
The following directories and files have been created or updated:  
-----  
• /Users/selwyn/Sites/d9book2/web/modules/custom/general/general.routing.yml  
• /Users/selwyn/Sites/d9book2/web/modules/custom/general/src/Controller/ExampleController.php
```

The file `general.routing.yml` will then contain:

```
general.book_example:
path: '/general/book_example'
defaults:
  _title: 'Book Example'
  _controller: '\Drupal\general\Controller\ExampleController::build'
requirements:
  _permission: 'access content'
```

The ExampleController.php file has these contents:

```
<?php

namespace Drupal\general\Controller;

use Drupal\Core\Controller\ControllerBase;

/**
 * Returns responses for General routes.
 */
class ExampleController extends ControllerBase {

  /**
   * Builds the response.
   */
  public function build() {

    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t('It works!'),
    ];
  }

  return $build;
}

}
```

This is a huge timesaver!

Finding routes with Drush

Drush lets you figure out the controller associated with a route since version 10.5. Here are some of the options:

```
$ drush route
$ drush route --path=/user/1
$ drush route --name=update.status
$ sh route --url=https://example.com/node/1
```

more at https://www.drush.org/latest/commands/core_route/

All routes

Output from drush route. It lists the routes by name and the path they apply to.

```
$ drush route

'<button>': /
'<current>': /<current>
'<front>': /
'<nolink>': /
'<none>': /
admin_toolbar.run.cron: /run-cron
admin_toolbar.settings: /admin/config/user-interface/admin-toolbar
admin_toolbar_tools.cssjs: /admin/flush/cssjs
admin_toolbar_tools.flush: /admin/flush
admin_toolbar_tools.flush_menu: /admin/flush/menu
admin_toolbar_tools.flush_rendercache: /admin/flush/rendercache
admin_toolbar_tools.flush_static: /admin/flush/static-caches
admin_toolbar_tools.flush_twig: /admin/flush/twig
admin_toolbar_tools.flush_views: /admin/flush/views
admin_toolbar_tools.plugin: /admin/flush/plugin
admin_toolbar_tools.settings: /admin/config/user-interface/admin-toolbar-tools
admin_toolbar_tools.theme_rebuild: /admin/flush/theme_rebuild
batch_examples.batch: /batch-examples/batchform
...
...
```

Specific path

Output when checking a specific path

```
$ drush route --path /admin/flush

name: admin_toolbar_tools.flush
path: /admin/flush
defaults:
_controller: '\Drupal\admin_toolbar_tools\Controller\ToolbarController::flushAll'
_title: 'Flush all caches'
requirements:
_permission: 'administer site configuration'
 csrf_token: 'TRUE'
options:
_admin_route: true
_access_checks:
- access_check.permission
- access_check.csrf
```

Specific route name

```
$ drush route --name=admin_toolbar_tools.flush

name: admin_toolbar_tools.flush
path: /admin/flush
defaults:
_controller: '\Drupal\admin_toolbar_tools\Controller\ToolbarController::flushAll'
_title: 'Flush all caches'
requirements:
_permission: 'administer site configuration'
_csrf_token: 'TRUE'
options:
_admin_route: true
_access_checks:
- access_check.permission
- access_check.csrf
```

Getting some help from Chat GPT

At the time of this writing, [OpenAI](#) has a research release of its ChatGPT available. You could always ask it to generate a controller for you as I did. I was amazed by the result. I asked it to:

Generate a Drupal 10 controller which displays a table of temperatures from 0 to 100 degrees in both celsius and fahrenheit

To which it replied:

```
Here is a Drupal 10 controller that generates a table of temperatures from 0 to 100 degrees in both Celsius and Fahrenheit:
```

```

<?php

namespace Drupal\mymodule\Controller;

use Drupal\Core\Controller\ControllerBase;

class TemperatureController extends ControllerBase {

/**
 * Display a table of temperatures.
 */
public function displayTable() {
  $rows = [];
  for ($i = 0; $i <= 100; $i++) {
    $rows[] = [
      'celsius' => $i,
      'fahrenheit' => $this->celsiusToFahrenheit($i),
    ];
  }
  $header = [
    ['data' => $this->t('Celsius'), 'field' => 'celsius'],
    ['data' => $this->t('Fahrenheit'), 'field' => 'fahrenheit'],
  ];
  $build['table'] = [
    '#type' => 'table',
    '#header' => $header,
    '#rows' => $rows,
    '#empty' => $this->t('No temperatures to display.'),
  ];
  return $build;
}

/**
 * Convert a temperature from Celsius to Fahrenheit.
 *
 * @param float $celsius
 *   The temperature in Celsius.
 *
 * @return float
 *   The temperature in Fahrenheit.
 */
protected function celsiusToFahrenheit($celsius) {
  return ($celsius * 9 / 5) + 32;
}
}

```

followed by this thoughtful addition:

To use this controller, you would need to define a route in your module's routing file that points to the displayTable method. For example:

```
mymodule.temperature_table:  
path: '/temperature-table'  
defaults:  
  _controller: '\Drupal\mymodule\Controller\TemperatureController::displayTable'  
  _title: 'Temperature Table'  
requirements:  
  _permission: 'access content'
```

It concluded with:

This would define a route at the path /temperature-table that displays the table of temperatures when accessed.

Resources

- [Introductory Drupal routes and controllers example from Drupal documentation updated October 2022](#)
 - [Structure of Routes from Drupal documentation updated September 2022](#)
 - [SymfonyCasts.com excellent course in Drupal 8](#)
 - [Controllers article from Drupalize.me updated December 2021](#)
-
-

[Back to top](#)

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

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

Security

TABLE OF CONTENTS

- [Overview](#)
- [Sanitizing on output to avoid Cross Site Scripting \(XSS\) attacks](#)
- [.htaccess magic](#)
 - [Disallowing access to users coming from a domain](#)
 - [Blocking core Drupal pages](#)
 - [Blocking file resources from all but a handful of sites](#)
 - [Time-based blocks](#)
 - [Blocking HTTP commands](#)
 - [Block specific user agents](#)
 - [Block traffic from robot crawlers](#)
 - [Blocking hotlinks](#)
- [Reviewing Server logs](#)
 - [Find the most frequent IP addresses](#)
 - [Find the most frequent User Agents](#)
 - [Find the most frequent URLs visited on your site](#)
- [General overview of hardening your Drupal site](#)
- [API functions best practices](#)
- [checkPlain](#)
- [Html::escape](#)
- [Drupal.checkPlain\(\)](#)
- [Use the database abstraction layer to avoid SQL injection attacks](#)
- [CSRF access checking](#)
- [example.routing.yml](#)
 - [Resources](#)

views 35

Overview

Drupal is a highly secure platform mostly due to the tireless efforts of the [security team](#).

Sanitizing on output to avoid Cross Site Scripting (XSS) attacks

The Twig theme engine now auto escapes everything by default. That means, every string printed from a Twig template (e.g. anything between ``) gets automatically sanitized if no filters are used.

[See Filters - Modifying Variables In Twig Templates](#) for the Twig filters available in Drupal. Notably, watch out for the "raw" filter, which does not escape output. Only use this when you are certain the data is trusted.

When rendering attributes in Twig, make sure that you wrap them with double or single quotes. For example, class="{{ class }}" is safe while class={{ class }} is not safe.

In order to take advantage of Twig's automatic escaping (and avoid safe markup being escaped) ideally all HTML should be outputted from Twig templates.

.htaccess magic

Disallowing access to users coming from a domain

Requests from specific domains can be blocked by adding the following to .htaccess:

```
RewriteCond %{HTTP_REFERER} domain-name\.com [NC]
RewriteRule .* - [F]
```

For multiple domains, use something similar to the following:

```
RewriteCond %{HTTP_REFERER} domain-one\.com [NC,OR]
RewriteCond %{HTTP_REFERER} domain-two\.com RewriteRule .* - [F]
```

Blocking core Drupal pages

Files such as CHangelog.txt can be used to quickly identify security vulnerabilities in your Drupal installation to a malicious script or user. While there are a number of ways to identify the version of Drupal that you are running, one quick addition to your .htaccess file can make it slightly less obvious.

```
# Various alias rules
Redirect 404 /CHANGELOG.txt
Redirect 404 /COPYRIGHT.txt
Redirect 404 /cron.php
Redirect 404 /INSTALL.mysql.txt
Redirect 404 /INSTALL.pgsql.txt
Redirect 404 /INSTALL.sqlite.txt
Redirect 404 /INSTALL.txt
Redirect 404 /install.php
Redirect 404 /LICENSE.txt
Redirect 404 /MAINTAINERS.txt
Redirect 404 /PATCHES.txt
Redirect 404 /README.txt
Redirect 404 /update.php
Redirect 404 /UPGRADE.txt
Redirect 404 /web.config
```

Blocking file resources from all but a handful of sites

You may want to keep a specific directory from being accessed by the general public, unless it's being pulled by a particular website. This example shows blocks requests to the /sites/default/files directory unless the request comes from www?, prod-kb, or the kb subdomains of example.com.

```
RewriteCond %{REQUEST_URI} ^/sites/default/files
RewriteCond %{HTTP_REFERER} !^http://prod-kb.example.com [NC]
RewriteCond %{HTTP_REFERER} !^http://kb.example.com [NC]
RewriteCond %{HTTP_REFERER} !^http://(www.)?example.com [NC] RewriteRule .* - [F]
```

Time-based blocks

If you are only allowed to expose your website for a specific time period, you can do that. This condition and rule blocks access until 4 PM.

```
RewriteCond %{TIME_HOUR} ^16$
RewriteRule ^.*$ - [F,L]
```

Blocking HTTP commands

You may not want to allow certain types of commands to be proceed by your site.

This blocks any HTTP request that is not a GET or a POST request.

```
RewriteCond %{REQUEST_METHOD} !(GET|POST)
RewriteRule .* - [F]
```

Block specific user agents

If your website is the victim of a DDoS attack, and you want to block a group of IP addresses using the same User Agent, the following code may be helpful. Replace the UserAgent with the name of the agent you want to block:

```
RewriteCond %{HTTP_USER_AGENT} UserAgent  
RewriteRule .* - [F,L]
```

You can also block more than one User Agent at a time with the [OR] ('or next condition') flag, and the [NC] ('no case') flag renders the string case insensitive. Here are some examples of some user-agents with properly escaped regexes:

```
RewriteCond %{HTTP_USER_AGENT} Baiduspider [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} HTTrack [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} Yandex [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} Scrapy [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} Mozilla/5.0\ \compatible;\ Yahoo [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} AppleNewsBot [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} Googlebot [NC,OR]  
RewriteCond %{HTTP_USER_AGENT} Mozilla/5.0\ \compatible;\ YandexBot [NC]  
RewriteRule .* - [F,L]  
Important
```

Properly escape characters inside your regex (regular expressions) to avoid website errors.

HTTP_USER_AGENT can use regex as an argument. As seen in the example above, many User Agents will require regex due to the complexity of their name. Rather than creating the rule manually, websites such as <https://www.regex-escape.com/regex-escaping-online.php> can help construct a properly-escaped regex quickly.

How to test that the block is working

Test that the site is responding:

```
curl -H "host:www.url_you_are_testing.url http://localhost/
```

Test that the user-agent (Pcore as an example) is indeed blocked:

```
curl -H "host:www.url_you_are_testing.url" -H "user-agent:Pcore-HTTP/v0.25.0" http://localhost.com/
```

Block traffic from robot crawlers

While a robot crawler may not technically be an attack, some crawlers can cause real problems. You can use this when the robots do not obey the robots.txt file, or if you need an immediate block, because robots.txt is generally not fetched immediately by crawlers.

```
RewriteCond %{HTTP_REFERER} ^$  
RewriteCond %{HTTP_USER_AGENT} "<exact_name_for_the_bot>"  
RewriteRule ^(.*)$ - [F,L]
```

Blocking hotlinks

The last thing most website owners want is other websites stealing their content, or worse - hotlinking to their images and stealing their bandwidth. Here's a simple bit of code that prevents it-modify domain.com to your domain name:

```
RewriteCond %{HTTP_REFERER} !$  
RewriteCond %{HTTP_REFERER} !http://(www.)?domain.com/ .*$ NC  
RewriteRule \.(gif|jpg|swf|flv|png)$ /feed/ R=302,L
```

Reviewing Server logs

Find the most frequent IP addresses

```
awk '{print $1}' access.log| sort | uniq -c | sort -nr| head
```

Find the most frequent User Agents

```
awk -F\" '{print $6}' access.log | sort | uniq -c | sort -nr | head
```

Find the most frequent URLs visited on your site

```
awk -F\" '{print $2}' access.log| awk -F? '{print $1}' | sort | uniq -c | sort -nr | head
```

This query strips the variables (anything after a question mark in your URL).

[Read more on Acquia.com](#)

General overview of hardening your Drupal site

[from Acquia.com](#)

Ensure up-to-date backups are safe and secure

- Initiate a production database backup
- Download a copy of recent database backups, and keep updated copies offsite
- If possible, also take backups of the file system

Ensure Drupal Core and Installed Modules are up to date Drupal Core updates often contain security patches. Outdated, unmaintained modules often contain known security vulnerabilities.

- Look for projects and modules covered by the Drupal Security Advisories
- Remove obsolete and unused modules
- Check for available updates under the Drupal admin console, or by using drush or composer.

Perform a user audit

- Ensure permissions are restricted and implemented correctly
- Remove any old or unneeded admin or privileged accounts

If a breach has occurred or internal threat, an attacker or internal threat may have added user(s) to retain access.

- Check for any new or unexpected user accounts

Password Checks

Bad passwords are the most common cause of site compromise.

- Ensure strong password requirements are enforced. A community contributed module that offers this functionality is Password Policy.
- Perform a check for bad passwords. A community contributed module that offers this functionality is [Drop the Ripper](#)

2-Factor Authentication

- Enforce 2-factor authentication (especially for admin and/or privileged accounts) to mitigate the threat of compromised passwords.
- **Review Site Functionality**
- Check that file uploads are restricted to intended file extension type (e.g. Do not allow .html uploads for an image)
- Ensure any sensitive data files are uploaded to secure directories only (e.g. Do not place personal data (PII) such as CVs or job applications in public 'files' directories)
- Review controls on web forms

Attackers will often target forms that generate outbound emails (e.g. "refer a friend" or "contact-us")

Try to keep messages generated from forms generic. Ensure CAPTCHA controls are used to prevent abuse

Web Application Firewall (WAF)

If a WAF is not already in place, Acquia strongly recommend implementing one.

[Acquia Cloud Edge Protect](#) is Acquia's WAF offering.

Edge Protect provides advanced security controls to restrict and block attacker traffic before it reaches the application stack. Common attack methods are identified and blocked automatically. WAFs are extremely effective in mitigating (D)DOS attacks.

API functions best practices

From [Writing secure code for Drupal](#)

- Use [t\(\)](#) and [Drupal::translation\(\)->formatPlural\(\)](#) with placeholders to construct safe, translatable strings. (See [Translation API overview](#) for more details.)
- Use [Html::escape\(\)](#) for plain text.
- Use [Xss::filter\(\)](#) for text that should allow some HTML tags.
- Use [Xss::filterAdmin\(\)](#) for text entered by admin users that should allow most HTML.
- Use [UrlHelper::stripDangerousProtocols\(\)](#) or [UrlHelper::filterBadProtocol\(\)](#) for checking URLs - the former can be used in conjunction with [SafeMarkup::format\(\)](#) - Oops, SafeMarkup was removed from Drupal 9. Rather use [FormattableMarkup](#)

Strings sanitized by [t\(\)](#), [Html::escape\(\)](#), [Xss::filter\(\)](#) OR [Xss::filterAdmin\(\)](#) are automatically marked safe, as are markup strings created from render arrays via [Renderer](#).

While it can also sanitize text, it's almost never correct to use [check_markup](#) in a theme or module except in the context of something like a text area with an associated text format.

checkPlain

This is from the file web/modules/custom/rsvp/src/Controller/ReportController.php

Here we pass [SafeMarkup::checkPlain`](#) to [array_map](#) to call it on each entry in the array. The entry array looks like:

```
$entry[name] = "fred";
$entry[email] = "fred@bloggs.com"
```

```
$rows = array_map('Drupal\Component\Utility\SafeMarkup::checkPlain', $entry);
```

Here is the whole function.

```
$rows = $this->getAllRSVPs();
foreach ($rows as $row) {
    // Sanitize each entry.
    $rows_array[] = array_map('Drupal\Component\Utility\SafeMarkup::checkPlain', $row);
}
```

Html::escape

If you have html like this: <script>alert(2)</script> which will be output as a mess with these sorts of characters: &, < etc. use [Html::escape](#) to avoid this.

```
$rows_array[] = array_map('Drupal\Component\Utility\Html::escape', $row);
```

Drupal.checkPlain()

It is best practice to use the server to sanitize text, but there may be situations where you need to lean on the client (browser) to provide additional or temporary sanitization, such as an HTML element that updates on the client side as the user enters text.

These examples are for use cases of outputting text to the DOM. If you're sending text to the server, such as when making an API call, you should review Back End practices.

You can use [Drupal.checkPlain`](#) to escape basic characters and prevent malicious elements being introduced into the DOM, avoiding some basic Clickjacking techniques.

Bad Practice:

```
var rawInputText = $("#form-input").text();
```

Good Practice:

```
var rawInputText = $('#form-input').text();
var escapedInputText = Drupal.checkPlain(rawInputText);
```

Use the database abstraction layer to avoid SQL injection attacks

Bad practice: Never concatenate data directly into SQL queries.

```
\Database::getConnection()->query('SELECT foo FROM {table} t WHERE t.name = '.$_GET['user']);
```

Good Practice:

Use proper argument substitution. The database layer works on top of PHP PDO, and uses an array of named placeholders:

```
\Database::getConnection()->query('SELECT foo FROM {table} t WHERE t.name = :name', [':name' => $_GET['user']]);
```

For a variable number of argument, use an array of arguments or use the select() method. See examples of each below:

```
$users = ['joe', 'poe', $_GET['user']];
\Database::getConnection()->query('SELECT f.bar FROM {foo} f WHERE f.bar IN (:users)', [':users' => $users]);
```

```
$users = ['joe', 'poe', $_GET['user']];
$result = \Database::getConnection()->select('foo', 'f')
->fields('f', ['bar'])
->condition('f.bar', $users)
->execute();
```

When forming a LIKE query, make sure that you escape condition values to ensure they don't contain wildcard characters like "%":

```
db_select('table', 't')
->condition('t.field', '%' . db_like($user), 'LIKE')
->execute();
```

Make sure that users cannot provide any operator to a query's condition. For example, this is unsafe:

```
db_select('table', 't')
->condition('t.field', $user, $user_input)
->execute();
```

Instead, set a list of allowed operators and only allow users to use those.

db_query, db_select, and db_like were deprecated and removed from Drupal 9 - instead you should use a database connection object and call the query, select, and [escapeLike](#) methods on it (the parameters are the same).

CSRF access checking

CSRF (Cross-Site Request Forgery) protection is now integrated into the routing access system and should be used for any URLs that perform actions or operations that do not use a form callback. In previous versions of Drupal, it was necessary to add a generated token as a query parameter to a URL and check this token manually in either the callback or the access callback. Now you can simply use the '_csrf_token' requirement on a route definition. Doing so will automatically add a token to the query string, and this token will be checked for you.

example.routing.yml

example:

```
path: '/example'
defaults:
  _controller: '\Drupal\example\Controller\ExampleController::content'
requirements:
  _csrf_token: 'TRUE'
```

Note that, in order for the token to be added, the link must be generated using the `url_generator` service via route name rather than as a manually constructed path.

```
$url = Url::fromRoute(
  'node_test.report',
  ['node' => $entity->id()],
  ['query' => [
    'token' => \Drupal::getContainer()->get('csrf_token')->get("node/{$entity->id()}/report")
  ]]);
```

[See API reference: CsrfTokenGenerator::get](#)

To validate token manually (e.g. without adding `_csrf_token: 'TRUE'` to your `mymodule.routing.yml` file) at the route destination you can use the token and value used for generating it.

```
// Validate $token from GET parameter.
\Drupal::getContainer()->get('csrf_token')->validate($token, "node/{$entity->id()}/report");
```

Note. regarding anonymous users. Currently the `_csrf_token` check fails for users without an active session, which includes most anonymous users. See: [#2730351: CSRF check always fails for users without a session](#)

Resources

- [Security on Drupal.org](#)
- [Blocking access using rewrites \(Acquia.com\)](#)
- [.htaccess documentation \(Acquia.com\)](#)
- [Ban module in Drupal core overview for blocking IP addresses \(Acquia.com\)](#)
- [Advanced Ban module](#)
- [Restricting website access \(Acquia.com\)](#)
- [Writing secure code for Drupal from Drupal.org update August 2022](#)
- [Twig Filters - Modifying Variables In Twig Templates](#)
- [Translation API overview on Drupal.org updated August 2022](#)
- [CSRF access checking on Drupal.org updated March 2023](#)

[Back to top](#)

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

Page last modified: Jul 31 2023.

[Edit this page on GitHub](#)

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

Services and Dependency Injection

TABLE OF CONTENTS

- [Overview](#)
- [Static](#)
- [Static Shorthand methods](#)
- [Services in action](#)
- [ControllerBase shortcuts](#)
- [Injected/Dependency Injection](#)
 - [Controller details](#)
 - [Controller Example 1](#)
 - [Controller Example 2](#)
- [Finding services](#)
- [Creating a custom service](#)
 - [Arguments](#)
 - [Passing the config factory to our service](#)
 - [Taxonomy Tree Custom Service](#)
- [Using your custom service](#)
- [Dependency Injection](#)
 - [Overview](#)
 - [Service Container](#)
 - [Controller Example 1](#)
 - [Controller Example 2](#)
 - [Blocks and other plugins](#)
- [Procedural to Class-based dependency injection](#)
- [Drush services commands](#)
 - [List all services](#)
 - [Generate custom service](#)
- [Resources](#)

views 137

Overview

Services provide a "decoupled" way to access classes and members. Services are pluggable and replaceable by registering them with a service container which makes them well suited to test with PHPUnit tests.

Services can be accessed using two possible methods: static and injected (using dependency injection).

Static

For .module files and classes which are not exposed to the service container (see below for a definition of the service container), you have to use the static method of retrieving the service:

```

// Access the Drupal JSON serialization service.

$this->jsonSerialization = \Drupal::service('serialization.json');
$this->jsonSerialization->decode($string);

// Access your custom service.

$this->salutationService = \Drupal::service('hello_world.salutation');
$this->salutationService->hello();

// Or.

$abc_retrieval_service = \Drupal::service('abc.aardvark_retrieval_service');
$aardvark_names = $abc_retrieval_service->getAardvarkNames();

```

You can also get the container and use that to get a service (and then use the service) e.g.:

```

$container = \Drupal::getContainer()
$this->keyValue = $container->get('keyvalue')->get($collection);

```

Static Shorthand methods

A few popular services also have shorthand methods in the core Drupal.php file for accessing them faster (and easier for IDE autocompletion), for example, \Drupal::entityTypeManager(). Check it out for services with shorthand methods:

e.g.

```

public static function routeMatch() {
  return static::getContainer()->get('current_route_match');
}

public static function currentUser() {
  return static::getContainer()->get('current_user');
}

public static function entityTypeManager() {
  return static::getContainer()->get('entity_type.manager');
}

public static function cache($bin = 'default') {
  return static::getContainer()->get('cache.' . $bin);
}

```

and many more..

Services in action

Most of these examples show Drupal configuration.

In this example we use the config.factory service (via the ::configFactory() shortcut) to change the system email plugin to use our mail plugin:

```

/*
 * \Drupal::configFactory() retrieves the configuration factory.
 *
 * This is mostly used to change the override settings on the configuration
 * factory. For example, changing the language, or turning all overrides on
 * or off.
 */

/**
 * Implements hook_install().
 */
function hello_world_install() {
  $config = \Drupal::configFactory()->getEditable('system.mail');
  $mail_plugins = $config->get('interface');
  if (in_array('hello_world', array_keys($mail_plugins))) {
    return;
  }
  $mail_plugins['hello_world'] = 'hello_world_mail';
  $config->set('interface', $mail_plugins);
  $config->save();
}

```

Here we use config.factory service to change some config variables:

```

$values = $form_state->getValues();
$address1 = $values['footer_address1'];
$address2 = $values['footer_address2'];

$config = \Drupal::configFactory()->getEditable('dat.header_footer_settings');
$config->set('footer_address1', $address1);
$config->set('footer_address2', $address2);
$config->save();

```

Here we use the config.factory service (via ::config() shorthand method) to load some values from Drupal config:

```

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

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

```

Get the email address for the site using the config.factory service:

```

// returns website@d9book.com
$to = \Drupal::configFactory()->getEditable('system.site')->get('mail');

```

ControllerBase shortcuts

ControllerBase.php comes prepackaged with functions to get the following services statically:

```
protected function entityTypeManager() {
protected function entityFormBuilder() {
protected function cache($bin = 'default') {
protected function config($name) {
protected function keyValue($collection) {
protected function state() {
protected function moduleHandler() {
protected function formBuilder() {
protected function currentUser() {
protected function languageManager() {
```

This allows quick access from within your controllers to these services if you need to do things like:

```
// Make an entityQuery
$storage = $this->entityTypeManager()->getStorage('node');
$query = $storage->getQuery();
$query
->accessCheck(TRUE)
->condition('type', 'article')
->condition('title', $name)
->count();
$count_nodes = $query->execute();

// Or.

// Get info about the current user.
$account = $this->currentUser();
$username = $account->getAccountName();
$uid = $account->id();
$message = "<br>Account info user id: " . $uid . " username: " . $username;
```

Injected/Dependency Injection

Using dependency injection is the preferred way to use services although it involves more steps. See the [Dependency Injection](#) section below for more details.

Using dependency injection requires that you create a constructor and a `create()` function in your controller class.

The `create` function gets the service container as a parameter and chooses the services it needs. The `create` function calls the constructor and passes the services as arguments and stores them as properties.

The process for a block (or plugin) is a little different:

Your block must implement `ContainerFactoryPluginInterface`. *Plugins only get access to the service container if they implement the `ContainerFactoryPluginInterface` e.g.*

```
class TestBlock extends BlockBase implements ContainerFactoryPluginInterface {
```

You must also add the extra parameters to the `create()` function and the constructor i.e. `$plugin_id` and `$plugin_definition` e.g.

```
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition)
```

Here is an example of a block constructor with the `AccountProxyInterface` parameter added so we can inject that service:

```
public function __construct(array $configuration, $plugin_id, $plugin_definition, AccountProxyInterface $account) {
parent::__construct($configuration, $plugin_id, $plugin_definition);
$this->account = $account;
}
```

More about services and dependency injection at <https://code.tutsplus.com/tutorials/drupal-8-properly-injecting-dependencies-using-di--cms->

[26314](#)

More about using dependency injection for blocks and other plugins <https://chromatichq.com/blog/dependency-injection-drupal-8-plugins>

Controller details

Here are the steps for implementing an injected service in a controller.

From: docroot/modules/custom/apitest/src/Controller/ApiTestController.php

1. Your controller must extend ControllerBase

```
class ApiTestController extends ControllerBase {
```

2. You need a protected variable to hold the service

```
/**  
 *  
 * The CmAPIClient.  
 *  
 * @var \Drupal\cm_api\CmAPIClient  
 */  
protected $cmAPIClient;
```

3. You need a create() function. This will get passed the \$container so it can call its get() member function to instantiate the service you need. This function then calls the constructor and passes its parameters to it.

```
public static function create(ContainerInterface $container) {  
    return new static(  
        $container->get('cm_api.client')  
    );  
}
```

Note, you can pass multiple services by adding additional \$container->get() calls like this:

```
public static function create(ContainerInterface $container) {  
    return new static(  
        $container->get('current_user'),  
        $container->get('path.current'),  
        $container->get('path.validator'),  
    );  
}
```

4. Your constructor will expect your newly instantiated service(s) as parameters:

```
/**  
 * ApiController constructor.  
 */  
public function __construct(CmAPIClient $cmAPIClient) {  
    $this->cmAPIClient = $cmAPIClient;  
}
```

Similarly, if you are getting multiple services, add the additional parameters to the constructor, as well as assigning the variables. E.g.

```
public function __construct(AccountProxyInterface $account, CurrentPathStack $path_stack, PathValidatorInterface $path_validator) {  
    $this->account = $account;  
    $this->pathStack = $path_stack;  
    $this->pathValidator = $path_validator;  
}
```

5. In your code, use the protected variable (`$cmAPIClient`) to call functions in the service:

```
$result = $this->cmAPIClient->catchAll('POST', $body);
```

Rejoice! Note. No need to make any routing changes. Drupal handles all the parameters with the instructions provided. etc.

Controller Example 1

Here is a complete controller which uses the `current_user` service:

```
<?php

namespace Drupal\di_examples\Controller;

use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class DiExamplesController extends ControllerBase {

  protected AccountProxyInterface $account;

  /**
   * Builds the response.
   */
  public function build() {

    $account = $this->account->getAccount();
    $username = $account->getAccountName();
    $uid = $account->id();

    $message = "<br>Account info user id: " . $uid . " username: " . $username;

    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t($message),
    ];
  }

  return $build;
}

public static function create(ContainerInterface $container) {
  return new static($container->get('current_user'));
}

public function __construct(AccountProxyInterface $account) {
  $this->account = $account;
}

}
```

Controller Example 2

This controller uses 3 different services:

```
<?php

namespace Drupal\di_examples\Controller;

use Drupal\Core\Controller\ControllerBase;
```

```

use Drupal\Core\Path\CurrentPathStack;
use Drupal\Core\Path\PathValidatorInterface;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Returns responses for DI Examples routes.
 */
class DiExamplesController extends ControllerBase {

  protected AccountProxyInterface $account;
  protected CurrentPathStack $pathStack;
  protected PathValidatorInterface $pathValidator;

  public static function create(ContainerInterface $container) {
    return new static(
      $container->get('current_user'),
      $container->get('path.current'),
      $container->get('path.validator'),
    );
  }

  public function __construct(AccountProxyInterface $account, CurrentPathStack $path_stack, PathValidatorInterface $path_validator) {
    $this->account = $account;
    $this->pathStack = $path_stack;
    $this->pathValidator = $path_validator;
  }

  /**
   * Builds the response.
   */
  public function build() {

    // Use the injected account.
    $account = $this->account->getAccount();

    // Use the ControllerBase static version.
    $account = $this->currentUser();

    $username = $account->getAccountName();
    $uid = $account->id();

    $message = "<br>Account info user id: " . $uid . " username: " . $username;

    $name = 'hello';

    // Use the ControllerBase static version to create an entityQuery.
    $storage = $this->entityTypeManager()->getStorage('node');
    $query = $storage->getQuery();
    $query
      ->condition('type', 'article')
      ->condition('title', $name)
      ->count();
    $count_nodes = $query->execute();
    $message .= "<br>Retrieved " . $count_nodes . " nodes";

    $path = $this->pathStack->getPath();
    $message .= "<br> Path: " . $path;
  }
}

```

```

$test_path = "/vote1";
$valid_path = $this->pathValidator->isValid($test_path);
$message .= "<br> Check for valid path: " . $test_path . " returned: " . $valid_path;

$build['content'] = [
  '#type' => 'item',
  '#markup' => $this->t($message),
];
}

return $build;
}
}

```

Finding services

Here is the process to find a commonly used service, the entityTypeManager which is used for entityQueries.

You can look at <https://api.drupal.org/api/drupal/services> and search for entity_type. This will result in:

See also

- [Services and Dependency Injection Container](#)
- [Service Tags](#)

Name contains	Tag contains	Apply	
entity_type			
Name	File	Class	Tags
entity_type.bundle.info	core/ core.services.yml	Drupal\Core\Entity\EntityTypeBundleInfo	
entity_type.listener	core/ core.services.yml	Drupal\Core\Entity\EntityTypeListener	
entity_type.manager	core/ core.services.yml	Drupal\Core\Entity\EntityTypeManager	plugin_manager_cache_clear
entity_type.repository	core/ core.services.yml	Drupal\Core\Entity\EntityTypeRepository	

Other projects

This tells you to that the service is in core.services.yml and that it is implemented in the EntityTypeManager class.

So in Drupal core's core.services.yml file you will find:

```

entity_type.manager:
  class: Drupal\Core\Entity\EntityTypeManager
  arguments: ['@container.namespaces', '@module_handler', '@cache.discovery', '@string_translation', '@class_resolver', '@entity.last_installed_schema.repository']
  parent: container trait
  tags:
    - { name: plugin_manager_cache_clear }

```

Looking in Drupal.php, you will also find a shorthand method:

```

/**
 * Retrieves the entity type manager.
 *
 * @return \Drupal\Core\Entity\EntityTypeManagerInterface
 *   The entity type manager.
 */
public static function entityTypeManager() {
  return static::getContainer()->get('entity_type.manager');
}

```

So to use this statically, you can use the following:

```

$storage = \Drupal::entityTypeManager()->getStorage($entity_type);
$query = $storage->getQuery();
$query = \Drupal::entityQuery('node')
  ->accessCheck(TRUE)
  ->condition('type', 'page')
  ->condition('status', 1);
$nids = $query->execute();

```

To do this using dependency injection you will need to inject `entity_type.manager`. Follow the procedure outlined above in Controller details.

Creating a custom service

To make a service, you will need two parts: a `module.services.yml` file and a controller.

In the `module.services.yml` file. You need a machine name for the service and a class that implements it. E.g. in `kitchen_product.services.yml` you might have the following:

```

services:
  kitchen_product.product_manager_service:
    class: Drupal\kitchen_product\ProductManagerService

```

Arguments

You can specify optional arguments to pass to your service which might be needed when Drupal instantiates your service. Here we pass the `keyvalue` service as well as a Boolean value:

```

services:
  highway.road_generator:
    class: Drupal\highway\RoadGenerator
  arguments:
    - '@keyvalue'
    - "%highway.road.use_key_value_cache%"

```

The `%highway.road.user_key_value_cache%` value is a special argument which has been defined above the `services` key as a parameter. Its value will default to true on production:

```

parameters:
  highway.road.use_key_value_cache: true

services:
  dino_roar.roar_generator:
    class: Drupal\dino_roar\Jurassic\RoarGenerator
  arguments:
    - '@keyvalue'
    - "%highway.road.use_key_value_cache%"

```

This means that it is configurable. If you need to set it to false during development, you can easily override it on your local machine. Simply

add it to the development.services.yml file like this:

```
# Local development services.  
#  
# To activate this feature, follow the instructions at the top of the  
# 'example.settings.local.php' file, which sits next to this file.  
parameters:  
highway.road.use_key_value_cache: false
```

Note you can also pass strings in the form 'blah' surrounded by single quotes.

Passing the config factory to our service

As shown above, arguments use the "arguments" key, which can have an array of services, each preceded by an @ symbol. Other values which are not services can also be passed. In the module.services.yml file below, we pass the config.factory service. You can find it in the core.services.yml file where you can see it maps to the Drupal\Core\Config\ConfigFactory class.

It looks like this in the core.services.yml:

```
config.factory:  
class: Drupal\Core\Config\ConfigFactory  
tags:  
- { name: event_subscriber }  
- { name: service_collector, tag: 'config.factory.override', call: addOverride }  
arguments: ['@config.storage', '@event_dispatcher', '@config.typed']
```

Here is how it looks in your module.services.yml:

```
services:  
hello_world.salutation:  
class: Drupal\hello_world\HelloWorldSalutation  
arguments: ['@config.factory']
```

Taxonomy Tree Custom Service

Here is an example where Daniel Sipos of <https://www.webomelette.com> creates a custom service to build a taxonomy tree. Read his article at <https://www.webomelette.com/loading-taxonomy-terms-tree-drupal-8>. The repo is at https://github.com/upchuk/taxonomy_tree. His code is reproduced below:

Here is the taxonomy_tree.services.yml file:

```
services:  
taxonomy_tree.taxonomy_term_tree:  
class: Drupal\taxonomy_tree\TaxonomyTermTree  
arguments: ['@entity_type.manager']
```

And the controller:

```
<?php  
  
namespace Drupal\taxonomy_tree;  
  
use Drupal\Core\Entity\EntityTypeManager;  
  
/**  
 * Loads taxonomy terms in a tree  
 */  
class TaxonomyTermTree {  
  
/**  
 * @var \Drupal\Core\Entity\EntityTypeManager
```

```

*/
protected $entityTypeManager;

/**
 * TaxonomyTermTree constructor.
 *
 * @param \Drupal\Core\Entity\EntityTypeManager $entityTypeManager
 */
public function __construct(EntityTypeManager $entityTypeManager) {
  $this->entityTypeManager = $entityTypeManager;
}

/**
 * Loads the tree of a vocabulary.
 *
 * @param string $vocabulary
 *   Machine name
 *
 * @return array
 */
public function load($vocabulary) {
  $terms = $this->entityTypeManager->getStorage('taxonomy_term')->loadTree($vocabulary);
  $tree = [];
  foreach ($terms as $tree_object) {
    $this->buildTree($tree, $tree_object, $vocabulary);
  }

  return $tree;
}

/**
 * Populates a tree array given a taxonomy term tree object.
 *
 * @param $tree
 * @param $object
 * @param $vocabulary
 */
protected function buildTree(&$tree, $object, $vocabulary) {
  if ($object->depth != 0) {
    return;
  }
  $tree[$object->tid] = $object;
  $tree[$object->tid]->children = [];
  $object_children = &$tree[$object->tid]->children;

  $children = $this->entityTypeManager->getStorage('taxonomy_term')->loadChildren($object->tid);
  if (!$children) {
    return;
  }

  $child_tree_objects = $this->entityTypeManager->getStorage('taxonomy_term')->loadTree($vocabulary, $object->tid);

  foreach ($children as $child) {
    foreach ($child_tree_objects as $child_tree_object) {
      if ($child_tree_object->tid == $child->id()) {
        $this->buildTree($object_children, $child_tree_object, $vocabulary);
      }
    }
  }
}

```

```
}
```

Using your custom service

This is identical to using a Drupal built in service. These are the steps:

- 1 In your controller, make sure your controller extends ControllerBase.
- 2 Add protected variable in your class to hold your service.
- 3 Add a `create()` function to get the service(s) from the service container
- 4 Add a constructor which stores a link to each service so you can call functions in those services.

Note. Follow the slightly different steps for injecting services into blocks when using your service for blocks or plugins.

Dependency Injection

Overview

Dependency injection is the practice of "injecting" services. A service is any object managed by the Drupal [Service container](#).

Drupal introduces the concept of services to decouple reusable functionality and makes these services pluggable and replaceable by registering them with a service container.

It is best practice to access any of the services provided by Drupal via the service container to ensure the decoupled nature of these systems is respected.

Services are used to perform operations like accessing the database or sending an e-mail. Rather than use PHP's native MySQL functions, we use the core-provided service via the service container to perform this operation so that our code can simply access the database without having to worry about whether the database is MySQL or SQLite, or if the mechanism for sending e-mail is SMTP or something else.

From <https://www.drupal.org/docs/drupal-apis/services-and-dependency-injection/services-and-dependency-injection-in-drupal-8>.

Service Container

The Service container is the PHP object which handles the instantiation of all required services. When you want to use a service, you ask the service container for one and then you can call methods on the service. The Drupal Service container is built on top of the [Symfony Service container](#).

More at <https://www.drupal.org/docs/drupal-apis/services-and-dependency-injection/services-and-dependency-injection-in-drupal-8>.

For example, in `core.services.yml`, you will find the `email.validator` service which references the `EmailValidator` class. This is what you will see in `core.services.yml`:

```
email.validator:  
  class: Drupal\Component\Utility\EmailValidator
```

Looking in the `EmailValidator.php` file, there is an `isValid()` function which you can call to validate email addresses. E.g.

```
$this->emailValidator->isValid()
```

Similarly, in `core.services.yml` there is a `current_route_match` service which references the class `CurrentRouteMatch`:

```
current_route_match:  
  class: Drupal\Core\Routing\CurrentRouteMatch  
  arguments: ['@request_stack']
```

Looking in `CurrentRouteMatch.php`, there is a `getRouteName()` function which can be used to get the current route name with a call like:

```
$this->currentRouteMatch->getRouteName()
```

Dig deeper in `core.services.yml` file for many more services.

Controller Example 1

Here is a complete controller which uses the current_user service:

```
<?php

namespace Drupal\di_examples\Controller;

use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class DiExamplesController extends ControllerBase {

  protected AccountProxyInterface $account;

  /**
   * Builds the response.
   */
  public function build() {

    $account = $this->account->getAccount();
    $username = $account->getAccountName();
    $uid = $account->id();

    $message = "<br>Account info user id: " . $uid . " username: " . $username;

    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t($message),
    ];
  }

  return $build;
}

public static function create(ContainerInterface $container) {
  return new static($container->get('current_user'));
}

public function __construct(AccountProxyInterface $account) {
  $this->account = $account;
}

}
```

Controller Example 2

This controller uses 3 different services

```
<?php

namespace Drupal\di_examples\Controller;

use Drupal\Core\Controller\ControllerBase;
use Drupal\Core\Path\CurrentPathStack;
use Drupal\Core\Path\PathValidatorInterface;
use Drupal\Core\Session\AccountProxyInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Returns responses for DI Examples routes.
 */
```

```

>Returns responses for DI Examples routes.

*/
class DiExamplesController extends ControllerBase {

    protected AccountProxyInterface $account;
    protected CurrentPathStack $pathStack;
    protected PathValidatorInterface $pathValidator;

    public static function create(ContainerInterface $container) {
        return new static(
            $container->get('current_user'),
            $container->get('path.current'),
            $container->get('path.validator'),
        );
    }

    public function __construct(AccountProxyInterface $account, CurrentPathStack $path_stack, PathValidatorInterface $path_validator) {
        $this->account = $account;
        $this->pathStack = $path_stack;
        $this->pathValidator = $path_validator;
    }

    /**
     * Builds the response.
     */
    public function build() {

        // Use the injected account.
        $account = $this->account->getAccount();

        // Use the ControllerBase static version.
        $account = $this->currentUser();

        $username = $account->getAccountName();
        $uid = $account->id();

        $message = "<br>Account info user id: " . $uid . " username: " . $username;

        $name = 'hello';

        // Use the ControllerBase static version to create an entityQuery.
        $storage = $this->entityTypeManager()->getStorage('node');
        $query = $storage->getQuery();
        $query
            ->accessCheck(TRUE)
            ->condition('type', 'article')
            ->condition('title', $name)
            ->count();
        $count_nodes = $query->execute();
        $message .= "<br>Retrieved " . $count_nodes . " nodes";

        $path = $this->pathStack->getPath();
        $message .= "<br> Path: " . $path;

        $test_path = '/vote1';
        $valid_path = $this->pathValidator->isValid($test_path);
        $message .= "<br> Check for valid path: " . $test_path . " returned: " . $valid_path;
    }
}

```

```

$build['content'] = [
  '#type' => 'item',
  '#markup' => $this->t($message),
];

return $build;
}

}

```

Blocks and other plugins

The process for a block (or plugin) is a little different:

Your block must implement `ContainerFactoryPluginInterface`. Plugins only get access to the service container if they implement the `ContainerFactoryPluginInterface` e.g.

```

class TestBlock extends BlockBase implements
ContainerFactoryPluginInterface {

```

You must also add the extra parameters to the `create()` and `__construct()` function i.e. `$plugin_id` and `$plugin_definition` e.g.

```
public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition)
```

Here is an example of a block constructor with the `AccountProxyInterface` parameter added as this is the service we want to inject:

```

public function __construct(array $configuration, $plugin_id, $plugin_definition, AccountProxyInterface $account) {
  parent::__construct($configuration, $plugin_id, $plugin_definition);
  $this->account = $account;
}

```

See more about using dependency injection for blocks and other plugins at <https://chromatichq.com/blog/dependency-injection-drupal-8-plugins>

Procedural to Class-based dependency injection

Here is the overview from the `Drupal.php` file for Drupal 9.5.0 of when and how to use dependency injection:

```

/**
 * Static Service Container wrapper.
 *
 * Generally, code in Drupal should accept its dependencies via either
 * constructor injection or setter method injection. However, there are cases,
 * particularly in legacy procedural code, where that is infeasible. This
 * class acts as a unified global accessor to arbitrary services within the
 * system in order to ease the transition from procedural code to injected OO
 * code.
 *
 * The container is built by the kernel and passed in to this class which stores
 * it statically. The container always contains the services from
 * \Drupal\Core\CoreServiceProvider, the service providers of enabled modules and any other
 * service providers defined in $GLOBALS['conf'][container_service_providers].
 *
 * This class exists only to support legacy code that cannot be dependency
 * injected. If your code needs it, consider refactoring it to be object
 * oriented, if possible. When this is not possible, for instance in the case of
 * hook implementations, and your code is more than a few non-reusable lines, it
 * is recommended to instantiate an object implementing the actual logic.
 *
 * @code

```

```

* // Legacy procedural code.

* function hook_do_stuff() {
*   $lock = lock()->acquire('stuff_lock');
*   // ...
* }

*
* // Correct procedural code.

* function hook_do_stuff() {
*   $lock = \Drupal::lock()->acquire('stuff_lock');
*   // ...
* }

*
* // The preferred way: dependency injected code.

* function hook_do_stuff() {
*   // Move the actual implementation to a class and instantiate it.
*   $instance = new StuffDoingClass(\Drupal::lock());
*   $instance->doStuff();
* }

*
* // Or, even better, rely on the service container to avoid hard coding a
* // specific interface implementation, so that the actual logic can be
* // swapped. This might not always make sense, but in general it is a good
* // practice.
* \Drupal::service('stuff.doing')->doStuff();
* }

*
* interface StuffDoingInterface {
*   public function doStuff();
* }

*
* class StuffDoingClass implements StuffDoingInterface {
*   protected $lockBackend;
*
*   public function __construct(LockBackendInterface $lock_backend) {
*     $this->lockBackend = $lock_backend;
*   }
*
*   public function doStuff() {
*     $lock = $this->lockBackend->acquire('stuff_lock');
*     // ...
*   }
* }
* @endcode
*
* @see \Drupal\Core\DrupalKernel
*/

```

Drush services commands

List all services

drush devel:services OR drush dcs OR

```
drush eval "print_r(\Drupal::getContainer()->getServiceIds());"
```

drush dcs | grep "PART OF SERVICE NAME" can find a service e.g. drush dcs | grep "access" will find all services with access in the name.

```
$ drush devel:services
- access_arguments_resolver_factory
- access_check.contact_personal
- access_check.cron
- access_check.csrf
- access_check.custom
- access_check.db_update
- access_check.default
- access_check.entity
- access_check.entity_bundles
- access_check.entity_create
- access_check.entity_create_any
- access_check.entity_delete_multiple
- access_check.field_ui.form_mode
- access_check.field_ui.view_mode
- access_check.header.csrf
- access_check.node.add
- access_check.node.preview
- access_check.node.revision
- access_check.permission
- access_check.quickedit.entity_field
- access_check.theme
- access_check.update.manager_access
- access_check.user.login_status
- access_check.user.register
- access_check.user.role
- access_manager
- account_switcher
- admin_toolbar_tools.helper
- ajax_response.attachments_processor
- ajax_response.subscriber
...
...
```

Generate custom service

Drush provides a great starting point by generating some useful code that you can easily build on. Consider using this facility as you write your code.

From https://www.drush.org/latest/generators/service_custom

drush generate service:custom. Generates a custom Drupal service

Also there are these gems:

drush generate service:logger. Generates a logger service

drush generate service:breadcrumb-builder. Generates a breadcrumb builder service

drush generate service:event-subscriber. Generates an event subscriber

drush generate service:middleware. Generates a middleware

drush generate service:param-converter. Generates a param converter service

drush generate service:path-processor. Generates a path processor service

drush generate service:request-policy. Generates a request policy service

drush generate service:response-policy. Generates a response policy service

drush generate service:route-subscriber. Generates a route subscriber

etc.

Resources

- [Services and Dependency Injection in the Drupal.org API documentation](#)
 - [Drupal 8: Properly injecting dependencies using DI by Danny Sipos from May 2016](#)
 - [Dependency injection in Drupal 8 plugins \(or blocks\) by Märt Matoo from March 2017](#)
 - [Drupal 8: Properly Injecting Dependencies Using DI by Danny Sipos May 2016](#)
 - [Inject a service in Drupal 8 showing an example of injecting http_client \(Guzzle\) by J M Olivas July 2015](#)
-
-

[Back to top](#)

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

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

State API, TempStore and UserData

TABLE OF CONTENTS

- [Overview](#)
- [State API](#)
 - [Using Drush to read the State API](#)
 - [Example accessing State API](#)
 - [Long strings broken into paragraphs](#)
- [UserData](#)
- [TempStore](#)
 - [PrivateTempStore](#)
 - [SharedTempStore](#)
 - [Injecting tempstore.shared](#)
- [Reference](#)

views 82

Overview

The State API, TempStore API, and UserData are all storage mechanisms in Drupal. There is some overlap in their capabilities however they can be used for different purposes. Here is a brief explanation of each:

State API: This provides a global way to store key-value pairs of data that need to persist between page loads or can be shared across different parts of the website. It is used to persist data such as cron key, last cron run, system last check for updates, installation time and whether the system is in maintenance mode. It is like configuration data except it can't be exported (and imported) and stored in source code, thereby making it a little more secure. Typically, configuration settings are exportable values used in modules, features, or installation profiles e.g. front page path.

UserData: This allows you to store user-specific data (in key-value pairs) in a similar manner as the State API. Because the data is specific to each user, it is useful for custom user preferences or other user-specific information.

TempStore: This provides a way to store user-specific data (also key-value pairs) that may be needed for a short period of time but does not need to be permanently stored. For example, it can be used to store data that is being edited in a form, allowing users to continue working on the form even if they navigate away from the page before saving the changes. It is also ideal for storing data such as the contents of a shopping cart. I've used tempstore to quickly access a list of nodes that need to be voted on by each voter. This means they don't need to run a complex set of queries more than once.

NOTE

Similarly to TempStore, you can also use the Drupal cache system to load complicated data really quickly. I use this for storing arrays of nodes and data for a complicated voting application to improve scalability and performance.

State API

The State API allows you to store small pieces of information specific to a site. This information is stored permanently in the database.

State API data is stored permanently in the `key_value` table. There is a `set()`, `get()` and `delete()` as well as `setMultiple()` and `getMultiple()` functions. The convention for the key name is to use periods(or full-stops) to separate words. E.g. `my.state.data` or `emergency.header.message`. Underscores are also used e.g. `system.cron_key` OR `system.cron_last`.

State settings are values which should usually not be exported to code, and only make sense in the context of one site. For example, cron key is a state setting whereas the front page path is a "config" variable.

To set a state value:

```
\Drupal::state()->set('system.cron_key', md5("This is an example of a bad cron key"));
```

In contrast, this is how you set a config value:

```
\Drupal::configFactory()
->getEditable('system.site')
->set('page.front', 'node/1')
->save();
```

In Drupal 7, this was done using `variable_set()`. If you are upgrading from a Drupal 7 site, check out this article from March 2021 about upgrading to the Drupal 8 State system at <https://www.drupal.org/node/1787318>

Writing state data looks like this:

```
\Drupal::state()->set('selwyn.important.string', 'abc');
\Drupal::state()->set('selwyn.more.important.string', 'def');
```

In the screenshot below, you can see where this is stored in the `key_value` table. Notice that the collection column is set to "state" indicating these are State API values.

collection	name	value
pathauto_state.taxonomy_term	7	i:1;
pathauto_state.user	2	i:0;
post_update	existing_updates	a:104:{i:0;s:44:"system_post_update_cl...
state	comment.maintain_entity_statistics	b:1;
state	comment.node_comment_statistics_scale	d:1;
state	field.field.deleted	a:0:{}
state	install_task	s:4:"done";
state	install_time	i:1609205427;
state	module_filter.recent	a:1:{s:14:"state_examples";i:1676125304;}
state	node.min_max_update_time	a:2:{s:11:"min_created";s:10:"1651362...";
state	router.path_roots	a:49:{i:0;s:5:"admin";i:1;s:8:"run-cron";...;
state	routing.menu_masks.router	a:43:{i:0;i:501;i:1;i:493;i:2;i:464;i:3;i:2...;
state	routing.non_admin_routes	a:254:{i:0;s:22:"admin_toolbar.run.cro...";
state	selwyn.important.string	s:3:"abc";
state	selwyn.more.important.string	s:3:"def";
state	system.cron_key	s:74:"h5uiERZajeMAK3Zdbpp3b69fUXb...";
state	system.cron_last	i:1676142747;
state	system.css_js_query_string	s:6:"rpxlsv";
state	system.maintenance_mode	b:0;
state	system.private_key	s:74:"h5uiERZajeMAK3Zdbpp3b69fUXb...";
state	system.profile.files	a:15:{s:10:"demo_umami";s:44:"core/p...";
state	system.theme.files	a:10:{s:6:"bartik";s:34:"core/themes/ba...";
state	twig_extension_hash_prefix	a:2:{s:19:"twig_extension_hash";s:43:"2...";
state	update.last_check	i:1676125163;
state	views.view_route_names	a:13:{s:13:"watchdog.page";s:14:"dblo...";
system.schema	admin_toolbar	i:8002;
system.schema	admin_toolbar_tools	i:8202;
		i:8000;

Using Drush to read the State API

You can use Drush to set, read or delete values in the State API: e.g. here we read the `selwyn1` key value:

```
$ drush ev 'return \Drupal::state()->get("selwyn1");'
abc
```

Example accessing State API

Here is code that shows examples of setting, getting and deleting state values:

```

<?php

namespace Drupal\state_examples\Controller;

use Drupal\Core\Controller\ControllerBase;

/**
 * Returns responses for State API, TempStore and UserData Examples routes.
 */
class StateExamplesController extends ControllerBase {

  /**
   * Builds the response.
   */
  public function build1() {

    // Set single value.

    \Drupal::state()->set('selwyn1', 'abc');
    \Drupal::state()->set('selwyn2', 'def');
    $build['content'][] = [
      '#type' => 'item',
      '#markup' => $this->t('Retrieved State API data: selwyn1: %selwyn1 and selwyn2: %selwyn2 .', [
        '%selwyn1' => \Drupal::state()->get('selwyn1'),
        '%selwyn2' => \Drupal::state()->get('selwyn2'),
      ]),
    ];
    // Delete single value.

    \Drupal::state()->delete('selwyn1');
    \Drupal::state()->delete('selwyn2');
    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t('Deleted (single) State API data.'),
    ];
    // Set multiple values.

    \Drupal::state()->setMultiple(['selwyn1' => 'ghi', 'selwyn2' => 'jkl']);
    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t('Retrieved State API data: selwyn1: %selwyn1 and selwyn2: %selwyn2 .', [
        '%selwyn1' => \Drupal::state()->get('selwyn1'),
        '%selwyn2' => \Drupal::state()->get('selwyn2'),
      ]),
    ];
    \Drupal::state()->deleteMultiple(['selwyn1', 'selwyn2']);
    $build['content'] = [
      '#type' => 'item',
      '#markup' => $this->t('Deleted (multiple) State API data.'),
    ];
    return $build;
  }
}

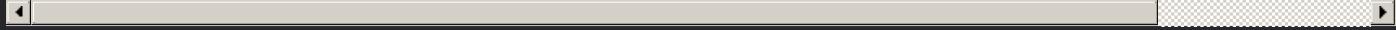
```

Long strings broken into paragraphs

If you store long strings, you can explode them into arrays of paragraphs for more control of their display.

```
$str = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque porttitor urna consequat dolor convallis iaculis. Nullam blandit ipsum eget odio semper rhoncus. Aenean libero tortor, ullamcorper vitae porttitor sit amet, posuere a ante. Vestibulum ipsum tellus, porta eu ligula at, porttitor congue nunc. Suspendisse felis lacus, tristique vel al \Drupal::state()->set('selwyn.long.string', $str);

// Explode string into array of paragraphs.
$test2 = explode( "\n", \Drupal::state()->get('selwyn.long.string'));
```



User Data

User Data is stored permanently in the `users_data` table. This is typically used for unstructured information pertaining to the user eg. user preferences, flags etc. I've used it to keep track of a user's progress in a complicated web application.

Here is an example where a value of `COMPLETED` is stored, read back and then deleted for the key `program.123.vote.0.finalized`:

```
public function build4() {

    /** @var \Drupal\user\UserDataInterface $userData */
    $userData = \Drupal::service('user.data');
    $user_id = 2;

    $userData->set('state_examples', $user_id, 'program.123.vote.0.finalized', 'COMPLETED');

    $build['content'][] = [
        '#type' => 'item',
        '#markup' => $this->t('Retrieved User Data API data: %value1.', [
            '%value1' => $userData->get('state_examples', $user_id, 'program.123.vote.0.finalized'),
        ]),
    ];

    $userData->delete('state_examples', $user_id, 'program.123.vote.0.finalized');

    return $build;
}
```

Here is the data stored in the `users_data` table. Notice that there is also some user data from the contact module as well as the data from the `state_examples` module that was created from the code above.

The screenshot shows the MySQL Workbench interface with the database set to 'db'. The 'Content' tab is selected, showing the 'users_data' table. The table has columns: uid, module, name, value, and serialized. There are two rows of data:

uid	module	name	value	serialized
2	contact	enabled	1	0
2	state_examples	program.123.vote.0.finalized	COMPLETED	0

TempStore

TempStore is used to keep temporary data across multiple requests. The data is intended to be non-cache data (i.e. not easily be rebuilt) and is stored in the `key_value_expire` table.

TempStore has two flavors, private and shared. The difference between them is that the private TempStore entries are connected to a specific user (via their user id) whereas shared TempStore entries can be shared between multiple users. Shared TempStore could for example be used to trigger a locking mechanism.

PrivateTempStore

The Drupal `tempstore.private` service is used to allow temporary user data that is available from one web request to the next. It is intended to be used for non-cache data that cannot easily be rebuilt. This includes work-in-progress data that isn't ready to be saved permanently.

The temporary store is a key/value store and can therefore store anything from a single Boolean value to a serialized object. (adapted from <https://www.hashbangcode.com/article/drupal-9-using-private-temporary-store-service>)

Stores and retrieves temporary data for a given owner.

A PrivateTempStore can be used to make temporary, non-cache data available across requests. The data for the PrivateTempStore is stored in one key/value collection. PrivateTempStore data expires automatically after a given timeframe.

By default, data is stored for one week (604800 seconds) before expiring.

The PrivateTempStore is different from a cache because the data in it is not yet saved permanently and so it cannot be rebuilt. Typically, the PrivateTempStore might be used to store work in progress that is later saved permanently elsewhere, e.g. autosave data, multistep forms, or in-progress changes to complex configuration that are not ready to be saved.

The PrivateTempStore differs from the SharedTempStore in that all keys are ensured to be unique for a particular user and users can never share data. If you want to be able to share data between users or use it for locking, use \Drupal\Core\TempStore\SharedTempStore.

(Adapted from

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21TempStore%21PrivateTempStore.php/class/PrivateTempStore/9.0.x>

In this example, we can see writing, reading, and deleting a private TempStore value:

```
public function build5() {

  // Private TempStore example.

  // Get the private TempStore for the state_examples module.
  $tempstore = \Drupal::service('tempstore.private')->get('state_examples');
  $tempstore->set('selwyn.important.string', 'abcdef');

  $build['content'][] = [
    '#type' => 'item',
    '#markup' => $this->t('Retrieved Private TempStore API data: %value1.', [
      '%value1' => $tempstore->get('selwyn.important.string'),
    ]),
  ];
}

// $tempstore->delete('selwyn.important.string');

return $build;
}
```

Drupal automatically inserts the user id (e.g. 1:) into the front of the name field so that the data is connected with that user. Notice in the database screenshot below that the name field (i.e. 1:selwyn.important.string) is preceded by 1: which makes it unique. Even though the code specified "selwyn.important.string" each user gets a unique copy. I was logged in as user 1 when I ran this program initially, and then I ran it again logged in as user 2.

The screenshot shows the MySQL Workbench interface with the 'Content' tab selected. In the left sidebar, under 'TABLES', the 'key_value' table is selected. The main pane displays the data in the 'key_value' table, which has columns: collection, name, value, and expire. The data shows multiple entries for the 'system' collection, each with a unique name starting with '1:' followed by the string 'selwyn.important.string'. Other entries include 'advisories_response', 'update_project_data', 'update_project_projects', 'admin_toolbar', 'ctools', and 'devel'. The 'expire' column shows various timestamps, indicating the expiration of the data.

collection	name	value	expire
system	advisories_response	a:0:{}	1676188997
tempstore.private.state_exa...	1:selwyn.important.string	O:8:"stdClass":3:{s:5:"owner";s:1...	1676775779
tempstore.private.state_exa...	2:selwyn.important.string	O:8:"stdClass":3:{s:5:"owner";s:1...	1676777078
tempstore.private.state_exa...	c6msZnneg8HJUsN9K9-Dl0gwHN...	O:8:"stdClass":3:{s:5:"owner";s:4...	1676777062
update	update_project_data	a:12:{s:6:"drupal";a:16:{s:4:"nam...	1676170997
update	update_project_projects	a:12:{s:6:"drupal";a:6:{s:4:"name"...	1676170997
update_available_releases	admin_toolbar	a:10:{s:5:"title";s:13:"Admin Tool...	1676211562
update_available_releases	ctools	a:10:{s:5:"title";s:25:"Chaos Tool...	1676211562
update_available_releases	devel	a:10:{s:5:"title";s:5:"Devel";s:10:"...	1676211562

In the next example we're storing an array with the key "selwyn.important.array". The array is automatically serialized into the value field. Incidentally, you don't have to specify the type of content (i.e. array) as I have. Instead of using selwyn.important.array, you could use selwyn.important.banana or selwyn.important.kiwi (depending on your fruit preference).

```
// Store an array.
$array = [
  'id' => '123',
  'name' => 'Dave',
];
$tempstore->set('selwyn.important.array', $array);
$array = $tempstore->get('selwyn.important.array');

$build['content'][] = [
  '#type' => 'item',
  '#markup' => $this->t('Retrieved Private TempStore API data: id = %value1, name= %value2.', [
    '%value1' => $array['id'],
    '%value2' => $array['name'],
  ]),
];

// Delete the data.
$tempstore->delete('selwyn.important.array');
```

db				Structure	Content	Relations	Triggers	Table Info	Query	Table History	Users	Console	
				Search: collection						Filter			
key_value													
TABLES													
key_value													
key_value_expire													
collection				name				value					
system				advisories_response				a:0:{}				1676188997	
tempstore.private.state_exa...	1:selwyn.important.array			O:8:"stdClass":3:{s:5:"owner";s:1...				O:8:"stdClass":3:{s:5:"owner";s:1...				1676777640	
tempstore.private.state_exa...	1:selwyn.important.string			O:8:"stdClass":3:{s:5:"owner";s:1...				O:8:"stdClass":3:{s:5:"owner";s:1...				1676777640	
tempstore.private.state_exa...	2:selwyn.important.string			O:8:"stdClass":3:{s:5:"owner";s:1...				O:8:"stdClass":3:{s:5:"owner";s:1...				1676777078	
tempstore.private.state_exa...	c6msZnnegBHJUsN9K9-Dl0gwHN...			O:8:"stdClass":3:{s:5:"owner";s:4...				O:8:"stdClass":3:{s:5:"owner";s:4...				1676777062	
update	update_project_data			a:12:{s:6:"drupal";a:16:{s:4:"nam...				a:12:{s:6:"drupal";a:16:{s:4:"nam...				1676170997	
update	update_project_projects			a:12:{s:6:"drupal";a:6:{s:4:"name"...				a:12:{s:6:"drupal";a:6:{s:4:"name"...				1676170997	
update_available_releases	admin_toolbar			a:10:{s:5:"title";s:13:"Admin Tool...				a:10:{s:5:"title";s:13:"Admin Tool..."				1676211562	
update_available_releases	ctools			a:10:{s:5:"title";s:25:"Chaos Tool..."				a:10:{s:5:"title";s:25:"Chaos Tool..."				1676211562	
update_available_releases	devel			a:10:{s:5:"title";s:5:"Devel";s:10:"...				a:10:{s:5:"title";s:5:"Devel";s:10:"..."				1676211562	
update_available_releases	drupal			a:9:{s:5:"title";s:11:"Drupal core",...				a:9:{s:5:"title";s:11:"Drupal core",...				1676211562	
update_available_releases	jquery_ui			a:10:{s:5:"title";s:9:"jQuery UI";s:...				a:10:{s:5:"title";s:9:"jQuery UI";s:...				1676211562	
update_available_releases	jquery_ui_autocomplete			a:10:{s:5:"title";s:22:"jQuery UI A..."				a:10:{s:5:"title";s:22:"jQuery UI A..."				1676211562	

This is a screenshot of the value for the array field above. Notice the id = 123 and name = Dave. There is no need to manipulate the data to get it stored away safely. Drupal will serialize it for you:

collection	name	value
system	advisories_response	a:0:{}
tempstore.private.state_exa...	1:selwyn.important.array	O:8:"stdClass":3:{s:5:"owner";s:1...

Field: "value" – LONGBLOB(4294967295) NOT NULL

```
O:8:"stdClass":3:{s:5:"owner";s:1:"1";s:4:"data";a:2:{s:2:"id";s:3:"123";s:4:"name";s:4:"Dave";}
s:7:"updated";i:1676172840;}
```

[Edit All Fields in Pop-up Sheet](#)

Also see this article showing how to save values from a form and then later retrieve and process them in a controller. Saving temporarily values of a form with Private Tempstore in Drupal 8 by Karim Boudjema Mar 2019 <http://karimboudjema.com/en/drupal/20190315/saving-temporary-values-form-private-tempstore-drupal-8>

SharedTempStore

Stores and retrieves temporary data for a given owner.

A SharedTempStore can be used to make temporary, non-cache data available across requests. The data for the SharedTempStore is stored in a key/value pair. SharedTempStore data expires automatically after a given timeframe.

By default, data is stored for one week (604800 seconds) before expiring.

The SharedTempStore is different from a cache because the data in it is not yet saved permanently and so it cannot be rebuilt. Typically, the SharedTempStore might be used to store work in progress that is later saved permanently elsewhere, e.g. autosave data, multistep forms, or in-progress changes to complex configuration that are not ready to be saved.

Each SharedTempStore belongs to a particular owner (e.g. a user, session, or process). Multiple owners may use the same key/value collection, and the owner is stored along with the key/value pair in the value field.

Every key is unique within the collection, so the SharedTempStore can check whether a particular key is already set by a different owner. This is useful for informing one owner that the data is already in use by another; for example, to let one user know that another user is in the process of editing certain data, or even to restrict other users from editing it at the same time. It is the responsibility of the implementation to decide when and whether one owner can use or update another owner's data.

If you want to be able to ensure that the data belongs to the current user, use \Drupal\Core\TempStore\PrivateTempStore. (Adapted from <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21TempStore%21PrivateTempStore.php/class/PrivateTempStore/9.0.x>)

Here is example code showing writing, reading and deleting data from SharedTempStore:

```
public function build6() {

  // Shared TempStore example.

  // Get the shared TempStore for the state_examples module.
  /** @var \Drupal\Core\TempStore\SharedTempStoreFactory $factory */
  $factory = \Drupal::service('tempstore.shared');
  $tempstore = $factory->get('state_examples');

  // Store an array.
  $agent_array = [
    'id' => '007',
    'name' => 'James Bond',
  ];
  $tempstore->set('selwyn.important.agent', $agent_array);

  // Retrieve the data.
  $array = $tempstore->get('selwyn.important.agent');
  $build['content'][] = [
    '#type' => 'item',
    '#markup' => $this->t('Retrieved Private TempStore API data: id = %value1, name= %value2.', [
      '%value1' => $array['id'],
      '%value2' => $array['name'],
    ]),
  ];
  $build['content'][] = [
    '#type' => 'text',
    '#value' => 'Data retrieved from SharedTempStore',
  ];

  // Delete the data.
  // $tempstore->delete('selwyn.important.agent');

  return $build;
}
```

And here is the data in the database:

Q key_value

Search: collection = Filter

TABLES

	collection	name	value	expire
key_value	system	advisories_response	a:0:{}	1676188997
key_value_expire	tempstore.private.state_examples	1:selwyn.important.string	O:8:"stdClass":3:{s:5:"owner";s:1:..."	1676779650
	tempstore.private.state_examples	2:selwyn.important.string	O:8:"stdClass":3:{s:5:"owner";s:1:..."	1676777078
	tempstore.private.state_examples	c6msZnnegBHJUsN9K9-DI0gwHN...	O:8:"stdClass":3:{s:5:"owner";s:4:..."	1676777062
	tempstore.shared.state_examples	selwyn.important.agent	O:8:"stdClass":3:{s:5:"owner";s:1:..."	1676779851
	tempstore.shared.state_examples	selwyn.important.other_value	O:8:"stdClass":3:{s:5:"owner";s:1:..."	1676779300
	tempstore.shared.state_examples	selwyn.important.value	O:8:"stdClass":3:{s:5:"owner";s:1:..."	1676779300
	update	update_project_data	a:12:{s:6:"drupal";a:16:{s:4:"nam..."	1676178099
	update	update_project_projects	a:12:{s:6:"drupal";a:6:{s:4:"name"..."	1676178099
	update_available_releases	admin_toolbar	a:10:{s:5:"title";s:13:"Admin Tool..."	1676211562
	update_available_releases	ctools	a:10:{s:5:"title";s:25:"Chaos Tool..."	1676211562
	update_available_releases	devel	a:10:{s:5:"title";s:5:"Devel";s:10:..."	1676211562
	update_available_releases	drupal	a:9:{s:5:"title";s:11:"Drupal core";...	1676211562
	update_available_releases	jquery_ui	a:10:{s:5:"title";s:9:"jQuery UI";s:..."	1676211562

And you can see the data owner in the screen shot below:

Field: "value" – LONGBLOB(4294967295) NOT NULL

O:8:"stdClass":3:{s:5:"owner";s:1:..."1";s:4:"data";a:2:{s:2:"id";s:3:"007";s:4:"name";s:10:"James Bond";}
s:7:"updated";i:167617501;}

owner of the data

user id 1

Edit All Fields in Pop-up Sheet

INJECTING TEMPSTORE.SHARED

If you want to inject the service rather than use it statically, you have to inject `tempstore.shared`. e.g. In the `module.services.yml` file below, we inject 3 services, including the `tempstore.shared`. Note that this is actually the `SharedTempStoreFactory` (and not the `SharedTempStore` class itself). Remember to derive the “collection” as shown in the php snippet below:

```
services:
  tea_teks_srp.vote_processor:
    class: Drupal\teateks_srp\VotingProcessor
    arguments: ['@entity_type.manager', '@current_user', '@tempstore.shared']
```

```
/** @var \Drupal\Core\TempStore\SharedTempStore|null */
protected ?SharedTempStore $tempStore = NULL;

public function __construct(EntityTypeManager $entityTypeManager, AccountInterface $account, SharedTempStoreFactory $sharedTempStoreFactory) {
  $this->entityTypeManager = $entityTypeManager;
  $this->account = $account;
  // Derive the collection.
  // Note. this will write into the key_value_expire table collection column: tempstore.shared.tea_teks_srp.  */
  $this->tempStore = $sharedTempStoreFactory->get('tea_teks_srp');
}

public function create(ContainerInterface $container) {
  return new static(
    $container->get('entity_type.manager'),
    $container->get('current_user'),
    $container->get('tempstore.shared')
  );
}
```

Here is an example of actually writing and then reading the value from the shared tempstore.

```
$now = new DrupalDateTime('now');
$votes[12345] = [
  'vote' => 'a',
  'reason' => '',
  'timestamp' => $now->format('Y-m-d H:i:s'),
  'serialized' => FALSE,
];
// Write.
$this->tempStore->set($this->tempstoreKey, $votes);
// Read.
$x = $this->tempStore->get($this->tempstoreKey);
```

In core.services.yml, you can see that tempstore.shared uses the SharedTempStoreFactory class:

```
tempstore.shared:
  class: Drupal\Core\TempStore\SharedTempStoreFactory
  arguments: ['@keyvalue.expirable', '@lock', '@request_stack', '@current_user', '%tempstore.expire%']
  tags:
    - { name: backend_overridable }
```

Reference

- [State API Overview updated September 2022](#)
- [If you are upgrading from a Drupal 7 site which uses variable_set, check out this article from March 2021 about upgrading to the Drupal 8 State system at] (<https://www.drupal.org/node/1787318>)
- [Drupal API documentation for UserData](#)
- [Drupal API documentation for UserData::get](#)
- [Storing user data such as preferences in Drupal 8 using the UserData service by Daniel Sipos Mar 2017](#)
- [Drupal Documentation on PrivateTempStore](#)
- [Drupal 8: Tempstore \(with code snippets\) by Oleksi Raiu](#)
- [Drupal 9: Using The Private Temporary Store Service by Phil Norton July 2022] (<https://www.hashbangcode.com/article/drupal-9-using-private-temporary-store-service>)
- [Saving temporarily values of a form with Private Tempstore in Drupal 8 by Karim Boudjema Mar 2019](#)

[Back to top](#)

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

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

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

Taxonomy

TABLE OF CONTENTS

- [Lookup term by name](#)
- [Lookup term name using its tid](#)
- [Lookup term using its uuid](#)
- [Load terms from a term reference field](#)
- [Find terms referenced in a paragraph in a term reference field](#)
- [Get URL alias from a term ID](#)
- [Load all terms for a vocabulary](#)
- [Load all terms for a vocabulary and put them in a select \(dropdown\)](#)
- [Create taxonomy term programatically](#)
- [Find all nodes with a matching term](#)
- [Find nodes with a matching term using entityQuery](#)

views 86

Lookup term by name

```

public function loadTermByName() {
  $vocabulary_id = 'event_category';
  $term_name = 'protest';
  $terms = \Drupal::entityTypeManager()
    ->getStorage('taxonomy_term')
    ->loadByProperties(['name' => $term_name]);

  if (empty($terms)) {
    $build['result'] = [
      '#type' => 'item',
      '#markup' => $this->t('No terms found'),
    ];
  }
  else {
    /** @var Term $term */
    foreach ($terms as $term) {
      $term_name = $term->getName();
      $build[$term_name] = [
        '#type' => 'item',
        '#markup' => $this->t('Term name: @term_name. Term id: @term_id', [
          '@term_name' => $term_name,
          '@term_id' => $term->id(),
        ]),
      ];
    }
  }

  return $build;
}

// Or the deprecated version.

$terms = taxonomy_term_load_multiple_by_name($term_name, 'opinion_categories');
if (empty($terms)) {
  $ra = [
    '#markup' => $this->t('Invalid term: @term', ['@term' => $category])
  ];
  return $ra;
}
// pop off the first one and grab it's term_id
$term = reset($terms);
$term_id = $term->get('tid')->value;

```

Lookup term name using its tid

```

use Drupal\taxonomy\Entity\Term;
$term = Term::load($term_id);

// OR

$term_name = \Drupal\taxonomy\Entity\Term::load($term_id)->label();

// Or

$term = \Drupal::entityTypeManager()->getStorage('taxonomy_term')->load($term_id);

if (empty($term)) {
  $render_array = [
    '#markup' => $this->t('Invalid term id: @termid', ['@termid' => $term_id])
  ];
} else {
  $term_name = $term->name;
  $render_array = [
    '#markup' => $this->t('Term name: @term_name', ['@term_name' => $term_name])
  ];
}

return $render_array;
}

```

Lookup term using its uuid

Each taxonomy term has a UUID. See the taxonomy_term_data table uuid field. We can load a taxonomy term by it's uuid as shown below:

Here we load a taxonomy term, get it's name and it's tid.

```

public function loadByUUID() {
  $uuid = 'd4a7bbc5-3b1b-46a4-bea4-01255365999f';

  $storage = \Drupal::entityTypeManager()->getStorage('taxonomy_term');
  $term_loaded_by_uuid = $storage->loadByProperties(['uuid' => $uuid]);
  $term = reset($term_loaded_by_uuid);
  $str = "Term not found";
  if (!empty($term)) {
    $term_name = $term->getName();
    $term_id = $term->id();
    $str = "Term uuid = $uuid<br>";
    $str .= "Term name = $term_name<br>";
    $str .= "Term id = $term_id<br>";
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];

  return $build;
}

```

Load terms from a term reference field

Retrieve the values in the field_event_category and display their term name, term id and their uuid. The call to referencedEntities() returns an array of term objects, so no need to call load() on them separately.

```
public function loadTermRef() {
  $nid = 24;
  $event_node = Node::load($nid);
  if ($event_node) {
    $categories = $event_node->get('field_event_category')->referencedEntities();
  }
  $str = "Terms:<br>";
  /**
   * @var \Drupal\Core\Entity\EntityInterface $category */
  foreach ($categories as $category) {
    $term_id = $category->id();
    $term_name = $category->getName();
    $uuid = $category->uuid();
    $str .= "Term name = $term_name<br>";
    $str .= "Term id = $term_id<br>";
    $str .= "Term uuid = $uuid<br>";
  }
  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
  return $build;
}
```

Find terms referenced in a paragraph in a term reference field

Loop thru all the instances of a paragraph reference and grab the term in the paragraph.

```
use Drupal\taxonomy\Entity\Term;
```

```
foreach ($node->get('field_my_para')->referencedEntities() as $ent){
  $term = Term::load($ent->$field_in_paragraph->target_id);
  $name = $term->getName();
  print_r($name);
}
```

Get URL alias from a term ID

This will return something like:

```
URL Alias for tid 3 = https://d9book2.ddev.site/category/rally
```

```

public function getTaxonomyAlias() {
  $tid = 3;
  //return taxonomy alias
  $options = ['absolute' => TRUE]; //FALSE will return relative path.

  // Build a URL.
  /** @var \Drupal\Core\Url $url */
  $url = Url::fromRoute('entity.taxonomy_term.canonical', ['taxonomy_term' => $tid], $options);
  $path_string = $url->toString();
  $str = "URL Alias for tid $tid = $path_string";

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $build;
}

```

Load all terms for a vocabulary

This code loads the terms into an array and displays them on screen. Note that you can't use id() or getName() on the objects returned from loadTree() as they are standard objects. If you load the actual term entities using Term::load(), then you can use entity functions like id() and getName(). It returns this:

```

Found 5 terms in vocabulary event_category
Term: Hunger strike term_id: 5
Term: Protest term_id: 4
Term: Rally term_id: 3
Term: Training term_id: 2
Term: Webinar term_id: 1

```

```

public function loadTerms() {
  $vocabulary_id = 'event_category';

  $terms = \Drupal::entityTypeManager()
    ->getStorage('taxonomy_term')
    ->loadTree($vocabulary_id);

  /** @var \Drupal\taxonomy\Entity\Term $term */
  foreach ($terms as $term) {
    $terms_array[] = [
      'id' => $term->tid,
      'name' => $term->name,
      'weight' => $term->weight,
    ];
  }
  $length = count($terms_array);
  $str = "Found $length terms in vocabulary $vocabulary_id<br>";

  foreach ($terms as $term) {
    $str .= 'Term: ' . $term->name . ' term_id: ' . $term->tid . '<br>';
  }
  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
  return $build;
}
}

```

Load all terms for a vocabulary and put them in a select (dropdown)

```

$vid = 'event_format';
$terms = \Drupal::entityTypeManager()->getStorage('taxonomy_term')->loadTree($vid);
foreach ($terms as $term) {
  $options[$term->tid] = $term->name;
}

$form['event_format']['active'] = [
  '#type' => 'radios',
  '#title' => $this->t('Event Format'),
  '#default_value' => 1,
  '#options' => $options,
//  '#options' => [
//    0 => $this->t('In-Person'),
//    1 => $this->t('Online'),
//    2 => $this->t('In-Person & Online'),
//  ],
];

```

Create taxonomy term programatically

Vid is the vocabulary id e.g. event_category or type

```

$term = Term::create([
  'name' => 'protest',
  'vid' => 'event_category',
])->save();

```

Find all nodes with a matching term

See the queries chapter for other ways to do this.

```
public function getMatchingNodes() {
  $term_id = 3;

  $nodes = \Drupal::entityTypeManager()->getStorage('node')->loadByProperties([
    'field_event_category' => $term_id,
  ]);

  $length = count($nodes);
  $str = "Found $length nodes matching term_id $term_id<br>";

  /** @var Node $node */
  foreach ($nodes as $node) {
    $str .= "Node " . $node->id() . ": Title: " . $node->getTitle() . "<br>";
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];
}

return $build;
}
```

Find nodes with a matching term using entityQuery

This finds the first 5 nodes that have the matching term.

```
protected function loadFirstOpinion($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->range(0, 5);
  $nids = $query->execute();
  $nodes = $storage->loadMultiple($nids);

  $ra = [];
  foreach ($nodes as $node) {
    $ra[] = [
      '#type' => 'markup',
      '#markup' => '<p>' . $node->getTitle(),
    ];
  }
  return $ra;
}
```

[Back to top](#)

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

Page last modified: Apr 14 2023.

[Edit this page on GitHub](#)



PHPUnit and Drupal Test Traits

TABLE OF CONTENTS

- [Overview](#)
- [Location of PHPUnit Tests](#)
- [Output files from running phpunit](#)
- [Setup PHPUnit tests](#)
- [Running PHPUnit tests in the DDEV container](#)
 - [Run PHPUnit test in DDEV from the host](#)
 - [Run PHPUnit test on the host](#)
- [Generate a unit test with drush](#)
- [My first PHPUnit test](#)
- [Drupal Test Traits Overview](#)
- [Install/setup Drupal Test Traits](#)
 - [What about testing browser interaction?](#)
- [My first DTT tests](#)
- [Running DTT tests](#)
 - [Run tests on the host](#)
 - [Run tests in a specific file](#)
 - [Run a specific test in a file](#)
- [Logging Test Output](#)
 - [Capture every page loaded](#)
 - [Capture an HTML page](#)
 - [Example of capturing a page](#)
 - [Screenshot using ExistingSiteSelenium2DriverTest](#)
- [Writing DTT Tests](#)
 - [Test locations](#)
 - [Generate DTT tests with drush](#)
 - [Example tests](#)
 - [ExampleTest.php creating user term, article](#)
 - [VotingPageTest](#)
 - [ExampleLoginTest.php](#)
 - [Selenium2DriverTest](#)
 - [Login to your site](#)
 - [Create a new user and login as that user:](#)
 - [Create an admin user](#)
 - [Login as an existing user](#)
 - [Fill out a form](#)
 - [Parameter gotcha](#)
 - [Data Provider](#)
 - [Fill a queue and run a trait](#)
- [Mink](#)
 - [Checking page return code](#)
 - [Grab the text from the page](#)
 - [Current URL](#)
- [Load and parse a CSV file](#)
- [Adding DTT to an existing site](#)

- [Install DTT and dev requirements with:](#)
- [Create phpunit.xml file](#)
- [Create bootstrap-fast.php](#)
- [Add .phpunit.result.cache file to .gitignore](#)
- [Remove DTT and core-dev](#)
- [Hide deprecation notices](#)
- [Troubleshooting DTT Tests](#)
 - [Which test?](#)
 - [PolyfillAssertTrait not found](#)
 - [Class not found errors](#)
 - [var_dump, echo, print](#)
- [Using Xdebug and PHPStorm to debug DTT scripts](#)
- [Resources](#)
 - [General](#)
 - [Documentation](#)
 - [Testing setup](#)
 - [Mocking](#)

views 129

Overview

Tests are written as PHP classes. These are executed in the terminal (or optionally from within PhpStorm). Each test consists of code with calls to various assert functions to test if an expected result matches an actual result. So if a function should return true, you will usually `assertTrue($return_value)` or if a unit returns 5, you will `assertEquals(5, $return_value)`.

E.g.

```
self::assertTrue($return_value);
```

from <https://phpunit.readthedocs.io/en/9.5/writing-tests-for-phpunit.html>

```
<?php
declare(strict_types=1);

use PHPUnit\Framework\TestCase;

final class StackTest extends TestCase
{
    public function testPushAndPop(): void
    {
        $stack = [];
        $this->assertSame(0, count($stack));

        array_push($stack, 'foo');
        $this->assertSame('foo', $stack[count($stack)-1]);
        $this->assertSame(1, count($stack));

        $this->assertSame('foo', array_pop($stack));
        $this->assertSame(0, count($stack));
    }
}
```

In a Drupal context, there are 4 types of tests. From <https://www.drupal.org/docs/automated-testing/types-of-tests> :

- Unit: PHPUnit-based tests with minimal dependencies. Base class: `Drupal\Tests\UnitTestCase`. They must be clean plain PHP.

- Kernel: PHPUnit-based tests with a bootstrapped kernel, and a minimal number of extensions enabled. Base class: Drupal\KernelTests\KernelTestCase. More at <https://www.drupal.org/docs/automated-testing/phpunit-in-drupal/kerneltestbase>
- Functional: PHPUnit-based tests with a full booted Drupal instance. Base class: Drupal\Tests\BrowserTestBase.
- FunctionalJavascript: PHPUnit-based tests that use WebDriver to perform tests of Javascript and Ajax functionality in the browser. Base class: Drupal\FunctionalJavascriptTests\WebDriverTestBase.

In addition, the Drupal project has some tests for JavaScript, written in JavaScript, that use the [Nightwatch framework](#).

Drupal Test Traits (DTT) adds 2 more types of tests:

- ExistingSite which use the Drupal API and existing site. Base class: weitzman\DrupalTestTraits\ExistingSiteBase
- ExistingSiteJavascript for testing Javascript or AJAX using Selenium and Chromedriver. Base class: weitzman\DrupalTestTraits\ExistingSiteSelenium2DriverTestBase

There are many examples of tests in core and contributed modules.

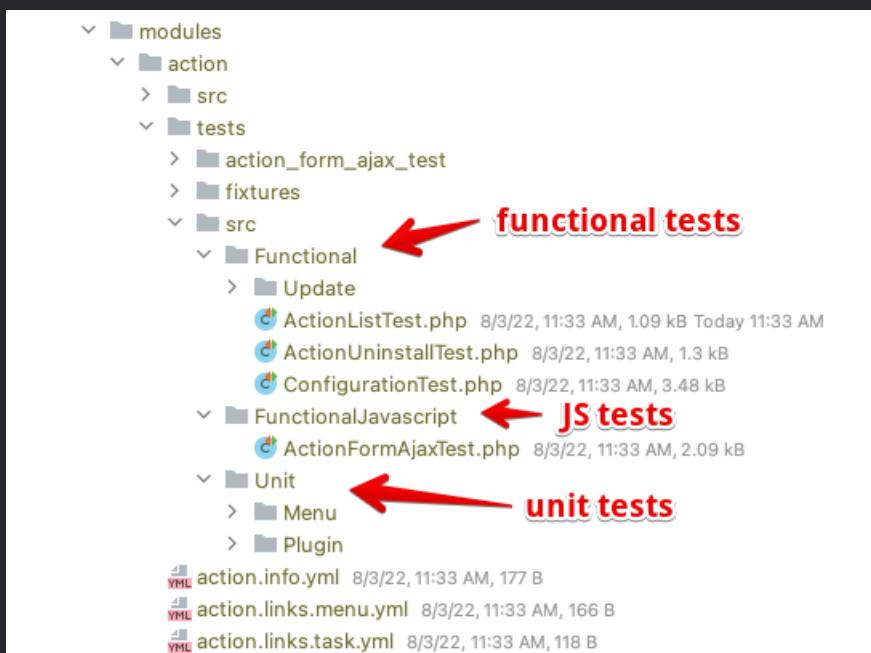
e.g. in docroot/modules/contrib/admin_toolbar/tests/src/Functional/ there are 3 functional tests: AdminToolbarAdminMenuTest.php, AdminToolbarAlterTest.php and AdminToolbarToolsSortTest.php

Every time a PHPUnit test is run, a fresh Drupal database and files are created. This guarantees that any existing data won't taint your test's outcomes. DTT bypasses this process and uses the existing site although it can clean up anything that is created in the test.

Location of PHPUnit Tests

Tests are always in the tests folder of modules. They are often in different folders e.g. tests/src/Unit, tests/src/Functional, tests/src/FunctionalJavascript

For core they are in web/core/modules (or docroot/core/modules) e.g. in the action module, the directory structure looks like this:

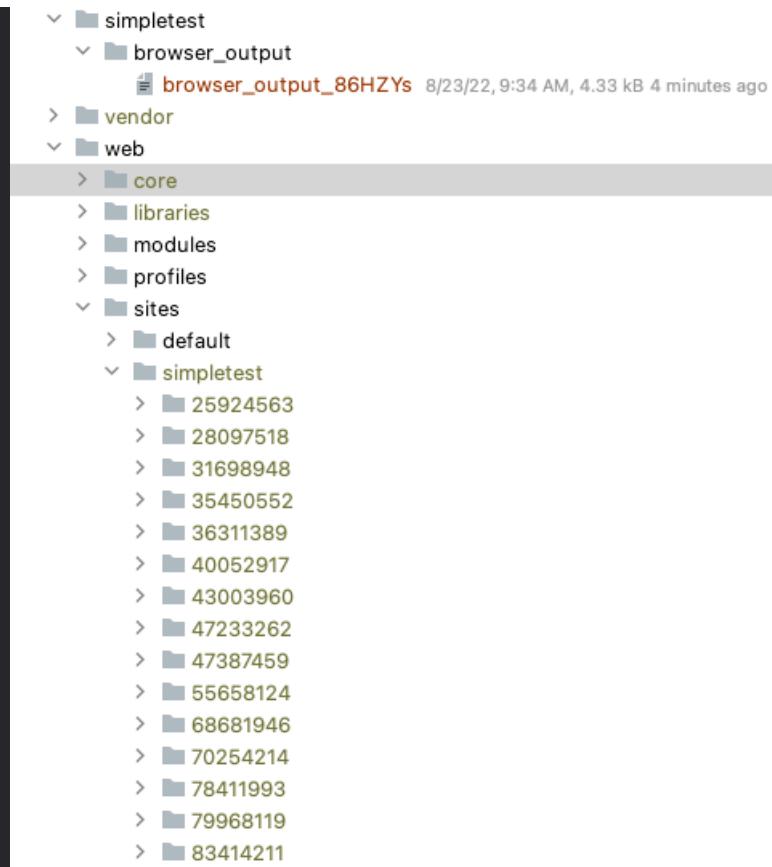


Output files from running phpunit

Depending on where you specified the output for your files in the phpunit.xml file e.g. in web/core/phpunit.xml I specified this path. Below you will see a bunch of directories (probably one for each run of the tests) in sites/simpletest/nnnnn

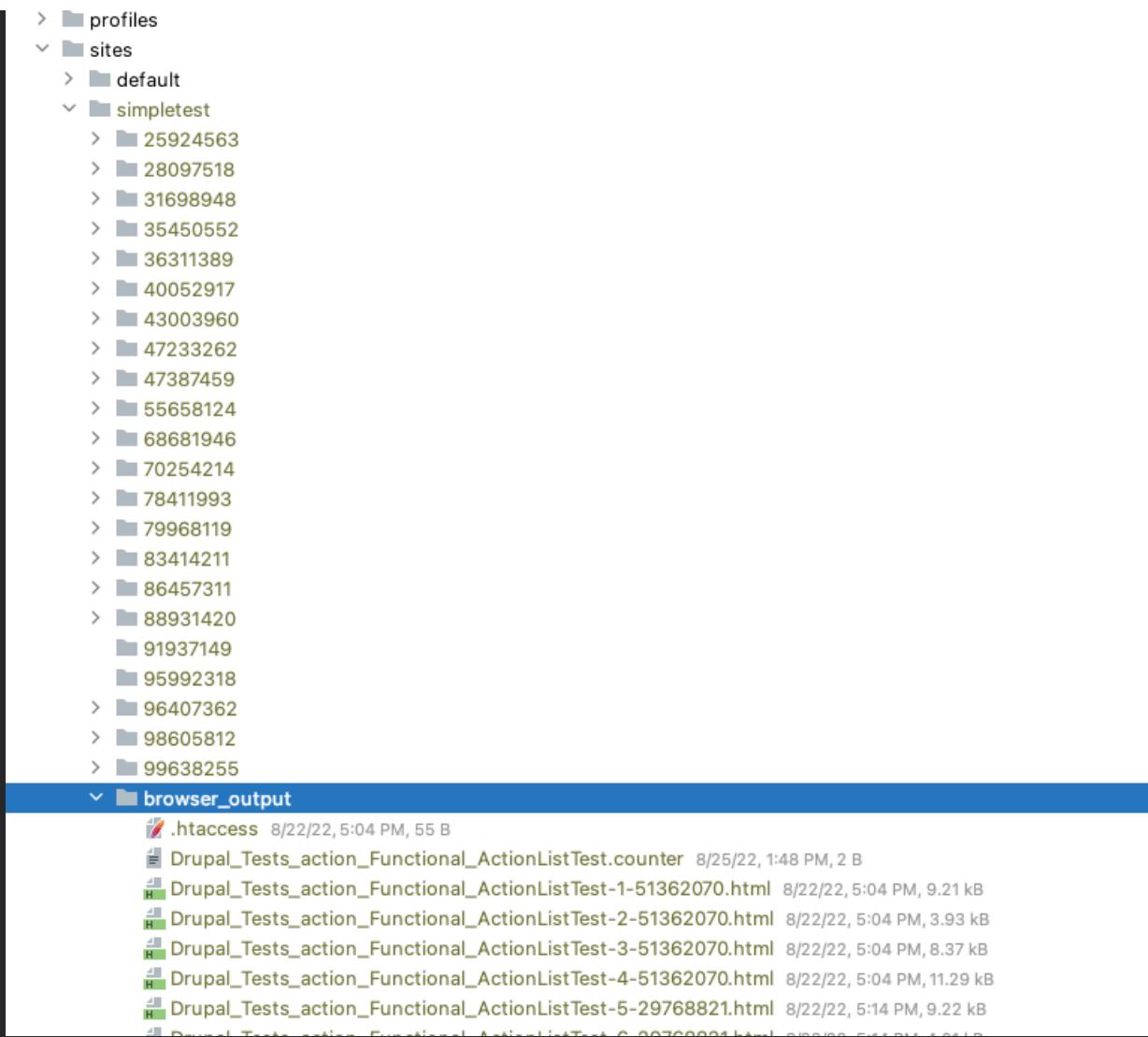
```

<!-- Example BROWSERTEST_OUTPUT_DIRECTORY value: /path/to/webroot/sites/simpletest/browser_output -->
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/var/www/html/simpletest/browser_output"/>
  
```

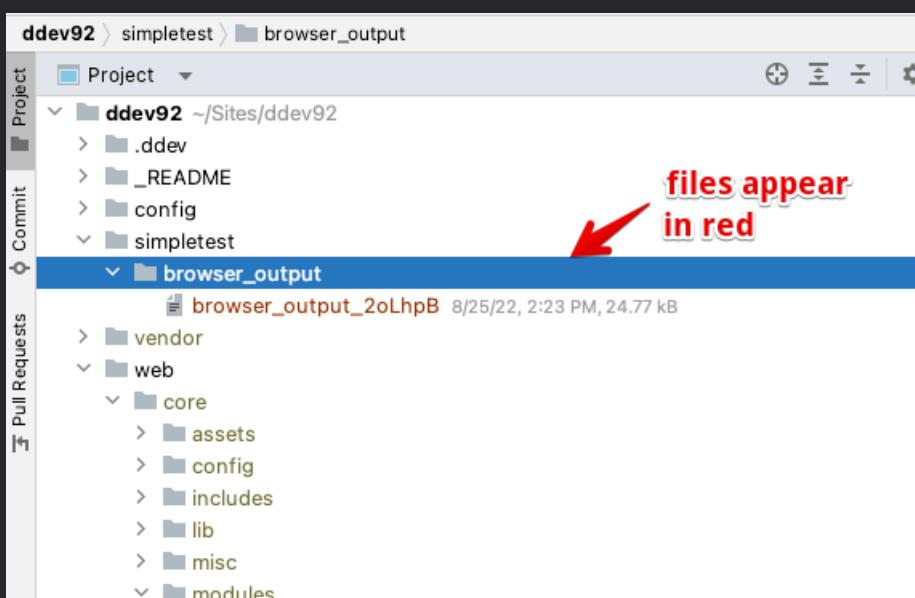


If you don't need to view the reports from your tests, you can safely delete these directories as well as the html files shown below.

At the same level is a browser_output directory which has some html files which reference the directories above:



While tests are running, I noticed that files appear in the `/simpletest/browser_output` folder at the topmost level of the project. They go away when the tests complete..



Setup PHPUnit tests

Using Drupal version 9.4.5 let's get PHPUnit tests running inside the ddev containers (not on the host machine).

To get started with phpunit, follow instructions at <https://www.drupal.org/docs/automated-testing/phpunit-in-drupal/running-phpunit-tests> (I updated this for ddev on 8-29-22)

Another useful reference is the web/core/tests/README.md that comes with Drupal.

If you installed using the drupal/recommended-project Composer template, development, you will need some dependencies which can be installed by requiring drupal/core-dev as a dependency in your project.

```
$ composer require drupal/core-dev --dev --update-with-all-dependencies
```

Install phpunit using:

```
$ composer require phpunit/phpunit
```

Also you might need prophecy for mocking:

```
$ composer require --dev phpspec/prophecy-phpunit:^2
```

You need a web/core/phpunit.xml so copy the one at web/core/phpunit.xml.dist.

Change the SIMPLETEST_BASE_URL, SIMPLETEST_DB AND BROWSERTEST_OUTPUT_DIRECTORY variables as shown below

```
<php>
<!-- Set error reporting to E_ALL. -->
<ini name="error_reporting" value="32767"/>
<!-- Do not limit the amount of memory tests take to run. -->
<ini name="memory_limit" value="-1"/>

<env name="SIMPLETEST_BASE_URL" value="http://localhost"/>
<env name="SIMPLETEST_DB" value="mysql://db:db@db/db"/>
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/var/www/html/simpletest/browser_output"/>
```

Refer to <https://mqlaman.dev/blog/running-drupals-phpunit-test-suites-ddev> for more info on using ddev describe to get suitable values for those variables

From web/core/tests/README.md

Copy the core/phpunit.xml.dist file to phpunit.xml, and place it somewhere convenient (inside the core directory may not be the best spot, since that directory may be managed by Composer or Git). You can use the -c option on the command line to tell PHPUnit where this file is (use the full path).

Settings to change in this file:

- SIMPLETEST_BASE_URL: The URL of your site
- SIMPLETEST_DB: The URL of your Drupal database
- The bootstrap attribute of the top-level phpunit tag, to take into account the location of the file
- BROWSERTEST_OUTPUT_DIRECTORY: Set to sites/simpletest/browser_output; you will also want to uncomment the printerClass attribute of the top-level phpunit tag.

Running PHPUnit tests in the DDEV container

This will run the unit tests included with the action module

```
$ ddev ssh
$ cd web
../vendor/bin/phpunit -c core/core/modules/action
```

The -c is a reference to the phpunit configuration file called phpunit.xml.

Another example test you could run is the book module:

You could also run `../vendor/bin/phpunit -c core core/modules/book` for a different set of tests. These tests are for the book module and take a very LONG time. (~30 minutes)

When running the tests for the action module, you should see:

```
PHPUnit 9.5.23 #StandWithUkraine

Warning: Your XML configuration validates against a deprecated schema.
Suggestion: Migrate your XML configuration using "--migrate-configuration"

Testing /var/www/html/web/core/modules/action
....S..
7 / 7 (100%)

Time: 03:25.204, Memory: 8.00 MB

OK, but incomplete, skipped, or risky tests!
Tests: 7, Assertions: 229, Skipped: 1.

HTML output was generated
http://localhost/sites/simpletest/browser_output/Drupal_Tests_action_Functional_ActionListTest-1-51362070.html
http://localhost/sites/simpletest/browser_output/Drupal_Tests_action_Functional_ActionListTest-2-51362070.html
http://localhost/sites/simpletest/browser_output/Drupal_Tests_action_Functional_ActionListTest-3-51362070.html
http://localhost/sites/simpletest/browser_output/Drupal_Tests_action_Functional_ActionListTest-4-51362070.html
http://localhost/sites/simpletest/browser_output/Drupal_Tests_action_Functional_ActionUninstallTest-1-80829434.html
```

Note in the output:S.., periods mean success, S means skipped, I means ignored, E means error.

You can choose to skip(S) or ignore(I) tests during development. This is done to speed up test runs or if you are mid-development and want to only run specific tests. Also these can be conditional such as when you only want a certain test to run if a condition is met e.g. a driver is present.

More details at <https://phpunit.readthedocs.io/en/9.5/textui.html>

Run PHPUnit test in DDEV from the host

You can also run the test in the containers from the host with:

```
$ ddev exec ./vendor/bin/phpunit -c web/core web/core/modules/action
```

Run PHPUnit test on the host

If you have installed PHP on your host computer, you should also be able to run Phpunit tests there with:

From the very top of the project:

```
$ vendor/bin/phpunit -c web/core web/core/modules/action
```

Generate a unit test with drush

You can use drush to generate a module e.g. drush gen module. (Just follow the prompts). Once you have a module, use drush gen test:unit for your module. Note. You need drush 11 for this.

Here is a generated file at `modules/custom/tea_teks_voting/tests/src/Unit/ExampleTest.php` with the following contents. It does not contain a real test; just a shell of a test which can run.

```

<?php

namespace Drupal\Tests\tea_teks_voting\Unit;

use Drupal\Tests\UnitTestCase;

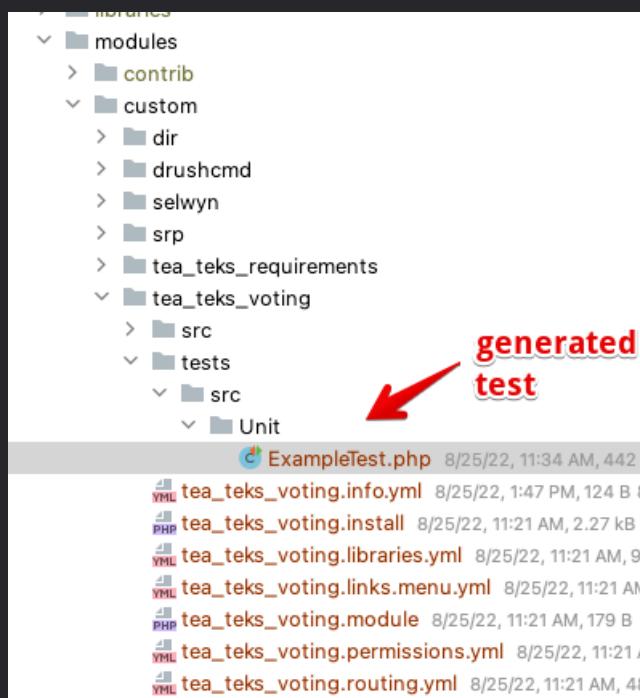
/**
 * Test description.
 *
 * @group tea_teks_voting
 */
class ExampleTest extends UnitTestCase {

/**
 * {@inheritDoc}
 */
protected function setUp(): void {
parent::setUp();
// @todo Mock required classes here.
}

/**
 * Tests something.
 */
public function testSomething() {
self::assertTrue(TRUE, 'This is TRUE!');
}
}

```

Drush wrote the test file in modules/custom/tea_teks_voting/tests/src/Unit/ExampleTest.php



To run this test use

```
$ drush ssh
```

```
selwyn@ddev92-web:/var/www/html $ vendor/bin/phpunit -c web/core/
web/modules/custom/tea_teks_voting/
```

You should see the following output:

```
PHPUnit 9.5.23 #StandWithUkraine

Warning: Your XML configuration validates against a deprecated schema.
Suggestion: Migrate your XML configuration using "--migrate-configuration"!

Testing /var/www/html/web/modules/custom/tea_teks_voting
    .
    1 / 1 (100%)

Time: 00:00.058, Memory: 4.00 MB

OK (1 test, 1 assertion)
```

Note. It does expect the file web/core/phpunit.xml to exist and be configured correctly. See setup above for details.

My first PHPUnit test

This is my test of a class I wrote called Requirements. It does require some Drupal to be bootstrapped, so it can't be a unit test. It must be a functional test and therefor in the modules/custom/tea_teks_requirements/tests/src/Functional directory.

```
<?php

namespace Drupal\Tests\tea_teks_requirements\Unit;

//use PHPUnit\Framework\TestCase;
use Drupal\Tests\UnitTestCase;
use Drupal\tea_teks_requirements\Requirements;

class RequirementsTest extends UnitTestCase {

    public function testEmptyRequirements() {
        $r = new Requirements();
        $empty = $r->isEmpty();
        $this->assertFalse($empty);
    }
}
```

The module must be enabled.

Copy web/core/phpunit.xml.dist to web/core/phpunit.xml

Edit the following lines of the file to look something like this:

```
<env name="SIMPLETEST_BASE_URL" value="http://localhost"/>
<env name="SIMPLETEST_DB" value="mysql://db:db@db/db"/>
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/var/www/html/simpletest/browser_output"/>
```

Run the test with

```
$ ddev ssh
$ cd web
$ ../../vendor/bin/phpunit -c core modules/custom/tea_teks_voting
```

You will see this output:

```
PHPUnit 9.5.23 #StandWithUkraine
```

```
Warning: Your XML configuration validates against a deprecated schema.  
Suggestion: Migrate your XML configuration using "--migrate-configuration"!
```

```
Testing /var/www/html/web/modules/custom/tea_teks_voting
```

```
1 / 1 (100%)
```

```
Time: 00:00.055, Memory: 4.00 MB
```

```
OK (1 test, 1 assertion)
```

Note. The OK means it worked.

Drupal Test Traits Overview

Drupal Test Traits is for testing Drupal sites that have content (versus Phpunit tests which start Drupal up, create an empty database and are consequently slower).

From <https://gitlab.com/weitzman/drupal-test-traits>:

[Behat](#) is great for facilitating conversations between business managers and developers. Those are useful conversations, but many organizations simply can't or won't converse via Gherkin. When you are on the hook for product quality and not conversations, this is a testing approach for you.

Before Drupal Test Traits, this framework was impossible to use without wiping the site's database after each test. DTT lets you keep your database and still test using the features of Drupal's BrowserTestBase and friends. See [DrupalTrait::setUp\(\)](#) for details (the bootstrap is inspired by [Drush](#)).

- Blog: [Introducing Drupal Test Traits](<https://medium.com/massgovdigital/introducing-drupal-test-traits-9fe09e84384c>)
- Blog: [Introducing Drupal Test Traits: Drupal extension for testing existing sites](<https://www.previousnext.com.au/blog/introducing-drupal-testing-traits-drupal-extension-testing-existing-sites>)
- Video: [Drupalcon presentation - Introducing Drupal Test Traits](<https://www.tag1consulting.com/blog/introducing-drupal-test-traits>).
- Repo: <https://gitlab.com/weitzman/drupal-test-traits>

DTT also supports [testing through a real browser using headless Chrome](#) or Selenium. So, testing client-side interactions like autocomplete, #states, viewports, and drag/drop is easy.

Like Drupal core, [DTT can save HTML snapshots for each URL that it navigates to](#). These files are very useful when debugging test failures.

Install/setup Drupal Test Traits

TLDR; You will need Drupal test traits installed with composer, possibly `drupal/core-dev` (also a composer install), a `/phpunit.xml` file, and a `/scripts/bootstrap-fast.php`. Add `weitzman/logintrait` with composer for adding users and logging in to your site. Finally for AJAX testing add a `docker-compose.testing.yaml` and using composer add `behat/mink-selenium2-driver`.

The details are as follows:

- 1 Install DTT. At the time of this writing the 1.6 version was out but there is a 2.x dev branch. Moshe recommends using that so use the following command to install it:

```
$ composer require --dev weitzman/drupal-test-trait:^2
```

- 1 Install the dev requirements with:

```
$ composer require drupal/core-dev --dev --update-with-all-dependencies
```

If you skip the `drupal/core-dev` step, you will see errors when you try to run the tests like this:

```
selwyn@tea3-web:/var/www/html$ ./vendor/bin/phpunit --bootstrap=./vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php  
./docroot/modules/custom/tea_teks/modules/tea_teks_requirements/tests/src/ExistingSite/RequirementsCreationTest.php
```

```
PHP Fatal error: Trait "Symfony\Bridge\PhpUnit\Legacy\PolyfillAssertTrait" not found in /var/www/html/docroot/sites/simpletest/Assert.php  
on line 91
```

```
Fatal error: Trait "Symfony\Bridge\PhpUnit\Legacy\PolyfillAssertTrait" not found in /var/www/html/docroot/sites/simpletest/Assert.php on  
line 91
```

- 1 Create a `phpunit.xml` file. I put `phpunit.xml` in the root of the project (not docroot or web.) See example file contents below.
- 2 Create a `bootstrap-fast.php`. I put `bootstrap-fast.php` in the `/scripts` (also in the root of the project). This file is included in DTT at `vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php`.
- 3 Add a `.ddev/docker-compose.testing.yaml` (don't accidentally type `docker-composeR` as Ddev won't create that container and you'll be left puzzled.)
- 4 Install the `mink-selenium2` driver with:\

```
$ composer require 'behat/mink-selenium2-driver' --dev
```

- 1 Install login traits with:

```
$ composer require weitzman/logintrait
```

Here are the `/phpunit.xml` file contents. Note the location of the `bootstrap-fast.php`:

```

<?xml version="1.0" encoding="UTF-8"?>

<!-- Copy the samples below into your own phpunit.xml file.-->

<!-- Using this project's bootstrap file allows tests in `ExistingSite`,
`ExistingSiteSelenium2DriverTest`, and `ExistingSiteWebDriverTest`
to run alongside core's test types. -->

<!-- If you use the default `bootstrap-fast.php` and get 'class not
found' errors while running tests, head over to
https://gitlab.com/weitzman/drupal-test-trait/-/blob/master/src/bootstrap-fast.php
for explanation on how to register those classes.
-->

<!-->
<!-<phpunit bootstrap="vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php">-->
<phpunit bootstrap="scripts/bootstrap-fast.php">
<php>
<!-->
<!--<env name="DTT_BASE_URL" value="https://ddev92.ddev.site"/>-->
<env name="DTT_BASE_URL" value="http://localhost"/>
<env name="DTT_API_URL" value="http://localhost:9222"/>
<!--<env name="DTT_MINK_DRIVER_ARGS" value="["chrome", {"chromeOptions": {"w3c": false}}, "http://localhost:4444/wd/hub"]"/>-->
<env name="DTT_MINK_DRIVER_ARGS" value="["firefox", null, "http://localhost:4444/wd/hub"]"/>
<env name="DTT_API_OPTIONS" value='{"socketTimeout": 360, "domWaitTimeout": 3600000}' />
<!-- Example BROWSETEST_OUTPUT_DIRECTORY value: /tmp
Specify a temporary directory for storing debug images and html documents.
These artifacts get copied to /sites/simpletest/browser_output by BrowserTestBase. -->
<env name="BROWSETEST_OUTPUT_DIRECTORY" value="" />
<!-- To disable deprecation testing completely uncomment the next line. -->
<!--<env name="SYMFONY_DEPRECATED_HELPER" value="disabled"/>-->
<!-- Specify the default directory screenshots should be placed. -->
<!--<env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />-->
<!-- Specify the default directory page captures should be placed.
When using the \Drupal\Tests\Listeners\HtmlOutputPrinter printerClass this will default to
/sites/simpletest/browser_output. If using another printer such as teamcity this must be defined.
-->
<!--<env name="DTT_HTML_OUTPUT_DIRECTORY" value="" />-->
</php>

<testsuites>
<testsuite name="unit">
<directory>./web/modules/custom/*/tests/src/Unit</directory>
<!--<directory>./web/profiles/custom/*/tests/src/Unit</directory>-->
</testsuite>
<testsuite name="kernel">
<directory>./web/modules/custom/*/tests/src/Kernel</directory>
<!--<directory>./web/profiles/custom/*/tests/src/Kernel</directory>-->
</testsuite>
<testsuite name="existing-site">
<!-- Assumes tests are namespaced as \Drupal\Tests\custom_foo\ExistingSite. -->
<directory>./web/modules/custom/*/tests/src/ExistingSite</directory>
<!--<directory>./web/profiles/custom/*/tests/src/ExistingSite</directory>-->
</testsuite>
<testsuite name="existing-site-javascript">
<!-- Assumes tests are namespaced as \Drupal\Tests\custom_foo\ExistingSiteJavascript. -->
<directory>./web/modules/custom/*/tests/src/ExistingSiteJavascript</directory>
<!--<directory>./web/profiles/custom/*/tests/src/ExistingSiteJavascript</directory>-->
</testsuite>
</testsuites>
</phpunit>

```

Here are the /scripts/bootstrap-fast.php contents from vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php:

```
<?php
/**
 * @file
 *
 * A bootstrap file for `phpunit` test runner.
 *
 * This bootstrap file from DTT is fast and customizable.
 *
 * If you get 'class not found' errors while running tests, you should copy this
 * file to a location inside your code-base --such as `/scripts`. Then add the
 * missing namespaces to the bottom of the copied field. Specify your custom
 * `bootstrap-fast.php` file as the bootstrap in `phpunit.xml`.
 *
 * Alternatively, use the `bootstrap.php` file, in this same directory, which is
 * slower but registers all the namespaces that Drupal tests expect.
 */

use Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter;
use weitzmanDrupalTestTraits\AddPsr4;

list($finder, $class_loader) = AddPsr4::add();
$root = $finder->getDrupalRoot();

// So that test cases may be simultaneously compatible with multiple major versions of PHPUnit.
$class_loader->addPsr4('Drupal\TestTools\\', "$root/core/tests");
if (class_exists('Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter')) {
  ClassWriter::mutateTestBase($class_loader);
}

// Register more namespaces, as needed.
# $class_loader->addPsr4('Drupal\Tests\my_module\\', "$root/modules/custom/my_module/tests/src");
```

Here is the docker-compose.testing.yaml from [Michael Strelan on drupal.org](#). Once you add this and restart DDEV, you will be able to do AJAX and Javascript testing of DTT tests. Note this is a "just works" situation as Michael puts it. Note that if you have this file in place, you don't need to provide all the env values in the phpunit.xml above.

```

version: '3.6'

services:
  web:
    links:
      - chromedriver:$DDEV_HOSTNAME
  environment:
    SYMFONY_DEPRECATED_HELPER: weak
  SIMPLETEST_DB_MYSQL: &simpletest_db_mysql mysql://db:db@db:3306/db
  SIMPLETEST_DB_SQLITE: &simpletest_db_sqlite sqlite://tmp/sites/simpletest/simpletest.db
  SIMPLETEST_DB: *simpletest_db_sqlite
  SIMPLETEST_BASE_URL: &base_url http://${DDEV_HOSTNAME}
  MINK_DRIVER_ARGS_WEBDRIVER: &mink_driver_args {"chrome": {"browserName": "chrome", "goog:chromeOptions": {"args": ["--disable-gpu", "--headless", "--no-sandbox"]}}, "selenium": {"browserName": "chrome", "goog:seleniumOptions": {"args": ["--headless", "--no-sandbox"]}}}
  BROWSETEST_OUTPUT_DIRECTORY: /var/www/html/web/sites/simpletest/browser_output
  BROWSETEST_OUTPUT_BASE_URL: ${DDEV_PRIMARY_URL}
  DTT_BASE_URL: *base_url
  DTT_MINK_DRIVER_ARGS: *mink_driver_args

chromedriver:
  image: drupalci/webdriver-chromedriver:production
  container_name: ddev-${DDEV_SITENAME}-chromedriver
  labels:
    com.ddev.site-name: ${DDEV_SITENAME}
    com.ddev.approot: ${DDEV_APPROOT}
  external_links:
    - ddev-router:${DDEV_SITENAME}.${DDEV_TLD}
  networks: [default, ddev_default]

```



What about testing browser interaction?

Without the docker-compose.testing.yaml, you can only run the most basic DTT tests such as [ExampleTest.php](#).

The lack of a headless Chrome container, will cause the [ExampleWebDriverTest.php](#) and [ExampleSeleniumDriverTest.php](#) to fail with errors like this when you try to run tests:

```
Error: Class "DMore\ChromeDriver\ChromeDriver" not found
```

Using `behat/mink-selenium2-driver`, you can run DTT against Chrome, Firefox or Edge (in WebDriver mode.) This setup also allows us to run Drupal core JS testing using Nightwatch. Although DTT supports browser testing using headless Chrome or Selenium. This facility is deprecated.

From <https://gitlab.com/weitzman/drupal-test-trait>:

(deprecated) `ExistingSiteWebDriverTest`. See [ExampleWebDriverTest.php](#). These tests make use of a headless Chrome browser but using [Chrome's Debugger Protocol](#). They are suited to testing Ajax and similar client side interactions. These tests run slower than `ExistingSite`. To use this test type you need to composer require 'dmore/chrome-mink-driver' --dev. Tests of this type should be placed in `tests/src/ExistingSiteJavascript`. Contrary to its name, this test type does not use the WebDriver protocol at all.

My first DTT tests

Tests must be in the `ExistingSite` directory and be namespaced as `Drupal\Tests\module_name\ExistingSite`.

Create `modules/custom/tea_teks_requirements/tests/src/ExistingSite/RequirementsCreationTest.php` with the following contents. This has 2 tests. Notice that we can use Drupal API calls just like when we write modules. This test is provided as an example of a simple test so you can see what they look like in real life.

```

<?php

namespace Drupal\Tests\tea_teks_requirements\ExistingSite;

use Drupal\node\Entity\Node;
use Drupal\tea_teks_requirements\Requirements;
use weitzman\Drupal\TestTraits\ExistingSiteBase;

class RequirementsCreationTest extends ExistingSiteBase {

  public function testEmptyRequirements() {
    $r = new Requirements();
    $empty = $r->isEmpty();
    $this->assertTrue($empty);
  }

  public function testLoadingRequirementsFromStandard() {
    $teks_standard_node = Node::load(8);
    $r = new Requirements($teks_standard_node);
    $empty = $r->isEmpty();
    $this->assertFalse($empty);
  }
}

```

Run it with:

```

$ ddev ssh

$ vendor/bin/phpunit --bootstrap=./vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php ./web/modules/custom/tea_teks_requirements/tests/src/ExistingSite/RequirementsCr

```

The output looks something like this:

```

PHPUnit 9.5.23 #StandWithUkraine

.. 2 / 2 (100%)

Time: 00:01.338, Memory: 16.00 MB

OK (2 tests, 2 assertions)

```

Along with a boatload of deprecation notices. (Note. we can hide these with `<env name="SYMFONY_DEPRECATIONS_HELPER" value="disabled"/>` in the `<php>` section of the `phpunit.xml` file):

```

Remaining direct deprecation notices (3)

1x: The "Symfony\Component\HttpFoundation\File\MimeType\MimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\MimeTypes" instead.

1x in RequirementsCreationTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: The "Symfony\Component\HttpFoundation\File\MimeType\FileBinaryMimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\FileBinaryMimeTypeGuesser" instead.

1x in RequirementsCreationTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: The "Symfony\Component\HttpFoundation\File\MimeType\FileInfoMimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\FileInfoMimeTypeGuesser" instead.

1x in RequirementsCreationTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

```

Remaining indirect deprecation notices (2)

1x: Return type of GuzzleHttp\Cookie\CookieJar::count() should either be compatible with Countable::count(): int, or the #
[!ReturnTypeWillChange] attribute should be used to temporarily suppress the notice

1x in RequirementsCreationTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: Return type of GuzzleHttp\Cookie\CookieJar::getIterator() should either be compatible with IteratorAggregate::getIterator():
Traversable, or the #![ReturnTypeWillChange] attribute should be used to temporarily suppress the notice

1x in RequirementsCreationTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

Running DTT tests

To run all the tests in the modules/custom/tea_teks_requirements directory, use the following:

```
$ ddev ssh  
  
$ vendor/bin/phpunit --bootstrap=./vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php modules/custom/tea_teks_requirements
```

The output for a successful run looks like this:

PHPUnit 9.5.23 #StandWithUkraine

. 1 / 1 (100%)

Time: 00:07.798, Memory: 20.00 MB

OK (1 test, 7 assertions)

Along with a boatload of deprecation notices. (Note. we can hide these with <env name="SYMFONY_DEPRECATED_HELPER" value="disabled"/> in the <php> section of the phpunit.xml file)

Remaining direct deprecation notices (3)

1x: The "Symfony\Component\HttpFoundation\File\MimeType\MimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\MimeTypes" instead.

1x in ExampleTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: The "Symfony\Component\HttpFoundation\File\MimeType\FileBinaryMimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\FileBinaryMimeTypeGuesser" instead.

1x in ExampleTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: The "Symfony\Component\HttpFoundation\File\MimeType\FileinfoMimeTypeGuesser" class is deprecated since Symfony 4.3, use
"Symfony\Component\Mime\FileinfoMimeTypeGuesser" instead.

1x in ExampleTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

Remaining indirect deprecation notices (3)

1x: Return type of GuzzleHttp\Cookie\CookieJar::count() should either be compatible with Countable::count(): int, or the #
[!ReturnTypeWillChange] attribute should be used to temporarily suppress the notice

1x in ExampleTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: Return type of GuzzleHttp\Cookie\CookieJar::getIterator() should either be compatible with IteratorAggregate::getIterator():
Traversable, or the #![ReturnTypeWillChange] attribute should be used to temporarily suppress the notice

1x in ExampleTest::setUp from Drupal\Tests\tea_teks_requirements\ExistingSite

1x: "Symfony\Component\DomCrawler\Crawler::text()" will normalize whitespaces by default in Symfony 5.0, set the second
"\$normalizeWhitespace" argument to false to retrieve the non-normalized version of the text.

1x in ExampleTest::testLlama from Drupal\Tests\tea_teks_requirements\ExistingSite

Once you've set up your bootstrap-fast.php file in /scripts, you can specify it in the phpunit.xml as

```
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
    backupGlobals="false"
    colors="true"
    bootstrap="scripts/bootstrap-fast.php"
    verbose="true"
    >
```

Then you don't need to pass it as a parameter. So running the test can look like this. Note no more reference to --bootstrap on the command line:

```
./vendor/bin/phpunit ./docroot/modules/custom/tea_teks/modules/tea_teks_voting/Tests/src/ExistingSite/RequirementsCreationTest.php
```

Or more simply:

```
vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/Tests/src/ExistingSite/RequirementsCreationTest.php
```

Also note you can specify the phpunit.xml file with -c parameter.

```
./vendor/bin/phpunit -c ./phpunit.xml ./docroot/modules/custom/tea_teks/modules/tea_teks_voting/Tests/src/ExistingSite/RequirementsCreationTest.php
```

Run tests on the host

You can use the same commands that you run in the DDEV containers if you have php 8.1 installed and running. They will run faster on the host. e.g.:

```
$ vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/TeamTest.php
```

Note. You can specify the location of bootstrap-fast.php in your /phpunit.xml. This file is found at vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php. Here a copy of bootstrap-fast.php is in the /scripts directory:

```
<?xml version="1.0" encoding="UTF-8"?>

<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
    backupGlobals="false"
    colors="true"
    bootstrap="scripts/bootstrap-fast.php"
    verbose="true"
    >
```

The source for bootstrap-fast.php is:

```

<?php
/**
 * @file
 * A bootstrap file for `phpunit` test runner.
 *
 * This bootstrap file from DTT is fast and customizable.
 *
 * If you get 'class not found' errors while running tests, you should copy this
 * file to a location inside your code-base --such as `/scripts`. Then add the
 * missing namespaces to the bottom of the copied field. Specify your custom
 * `bootstrap-fast.php` file as the bootstrap in `phpunit.xml`.
 *
 * Alternatively, use the `bootstrap.php` file, in this same directory, which is
 * slower but registers all the namespaces that Drupal tests expect.
 */

```

```

use Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter;
use weitzmanDrupalTestTraits\AddPsr4;

list($finder, $class_loader) = AddPsr4::add();
$root = $finder->getDrupalRoot();

// So that test cases may be simultaneously compatible with multiple major versions of PHPUnit.
$class_loader->addPsr4('Drupal\TestTools\\', "$root/core/tests");
if (class_exists('Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter')) {
  ClassWriter::mutateTestBase($class_loader);
}

// Register more namespaces, as needed.
# $class_loader->addPsr4('Drupal\Tests\my_module\\', "$root/modules/custom/my_module/tests/src");

```

Run tests in a specific file

You can be specific and only run all tests in a particular test file e.g.

```
$ vendor/bin/phpunit web/modules/custom/tea_teks_requirements/tests/src/ExistingSite/ExampleTest.php
```

Here is an example where the location of the bootstrap file is specified with --bootstrap:

```
$ ./vendor/bin/phpunit --bootstrap=./vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php ./web/modules/custom/tea_teks_requirements/tests/src/ExistingSite/ExampleTest.php
```

Run a specific test in a file

You can run a test that is called 'testVoter1Vote' in the VotingPageTest.php file with the following command. Note. If you have another test that starts with testVoter1Vote e.g. testVoter1VoteBlah, that test will be run also.

```
$ vendor/bin/phpunit --filter testVoter1Vote docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/VotingPageTest.php
```

Logging Test Output

PHPUnit can do all sorts of logging.

Capture every page loaded

For debugging, capturing all HTML requests can be useful.

At <https://gitlab.com/weitzman/drupal-test-trait/#debugging-tests> Moshe suggests:

- All HTML requests can be logged. To do so, add BROWSERTEST_OUTPUT_DIRECTORY=/tmp and --printer '\Drupal\Listeners\HtmlOutputPrinter' to the phpunit call. To disable deprecation notices, include SYMFONY_DEPRECATIONS_HELPER=disabled. Alternatively, you can specify these in your phpunit.xml ([example phpunit.xml](#)).

To add the printerclass to the phpunit.html see the printerClass line below:

```
<?xml version="1.0" encoding="UTF-8"?>

<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
    backupGlobals="false"
    colors="true"
    bootstrap="scripts/bootstrap-fast.php"
    verbose="true"
    printerClass="\Drupal\Tests\Listeners\HtmlOutputPrinter">

</php>
```

Putting this in the <php> section of the file causes all html requests to be output to /sites/simpletest/browser_output. I tried specifying a different directory with but it had no effect. Use this with caution (or only for debugging if you don't want to fill up hard drives.)

```
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
```

See the entire <php> section below:

```
<php>
<env name="DTT_BASE_URL" value="http://tea.ddev.site"/>
<env name="DTT_API_URL" value="http://chrome:9222"/>
<env name="DTT_MINK_DRIVER_ARGS" value='["chrome", {"browserName": "chrome", "chromeOptions": {"args": ["--disable-gpu", "--headless", "--no-sandbox"]}}, "http://chrome"]' />
<env name="DTT_API_OPTIONS" value='{"socketTimeout": 360, "domWaitTimeout": 3600000}' />
<!-- Example BROWSERTEST_OUTPUT_DIRECTORY value: /tmp
Specify a temporary directory for storing debug images and html documents.
These artifacts get copied to /sites/simpletest/browser_output by BrowserTestBase. -->
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
<!-- To disable deprecation testing completely uncomment the next line. -->
<env name="SYMFONY_DEPRECATIONS_HELPER" value="disabled"/>
<!-- Specify the default directory screenshots should be placed. -->
<!--<env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />-->
<!-- Specify the default directory page captures should be placed.

When using the \Drupal\Tests\Listeners\HtmlOutputPrinter printerClass this will default to
/sites/simpletest/browser_output. If using another printer such as teamcity this must be defined.

-->
<!--<env name="DTT_HTML_OUTPUT_DIRECTORY" value="" />-->
</php>
```

Capture an HTML page

From <https://gitlab.com/weitzman/drupal-test-trait#debugging-tests>:

- To write the current HTML of the page to a file, use \$this->capturePageContent(). If using HtmlOutputPrinter this will be saved to the browser_output directory. Alternatively you can specify DTT_HTML_OUTPUT_DIRECTORY=/path/to/output_directory which is required when using a different printer, such as Teamcity, which is enforced by PHPStorm.

Example of capturing a page

```

$url = Url::fromRoute('tea_teks_publisher.correlation_detail', [
  'node1' => $this->teaPublisherNid,
  'node2' => $this->testProgramOneNid,
  'node3' => $expectation_nid,
  'node4' => $correlation_nid,
]
);
$page_source = $this->drupalGet($url);
$this->capturePageContent();

```

Use the following setting to specify the output directory in the `phpunit.xml` file

```
<env name="DTT_HTML_OUTPUT_DIRECTORY" value="sites/simpletest/browser_output"/>
```

Here is the whole `<php>` section:

```

<php>
<env name="DTT_BASE_URL" value="http://tea.ddev.site"/>
<env name="DTT_API_URL" value="http://chrome:9222"/>
<env name="DTT_MINK_DRIVER_ARGS" value='["chrome", {"browserName": "chrome", "chromeOptions": {"args": ["--disable-gpu", "--headless", "--no-sandbox"]}}, "http://chrome"]' />
<env name="DTT_API_OPTIONS" value='{"socketTimeout": 360, "domWaitTimeout": 3600000}' />
<!-- Example BROWSERTEST_OUTPUT_DIRECTORY value: /tmp
      Specify a temporary directory for storing debug images and html documents.
      These artifacts get copied to /sites/simpletest/browser_output by BrowserTestBase. -->
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
<!-- To disable deprecation testing completely uncomment the next line. -->
<env name="SYMFONY_DEPRECATIONS_HELPER" value="disabled"/>
<!-- Specify the default directory screenshots should be placed. -->
<!--<env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />-->
<!-- Specify the default directory page captures should be placed.

      When using the \Drupal\Tests\Listeners\HtmlOutputPrinter printerClass this will default to
      /sites/simpletest/browser_output. If using another printer such as teamcity this must be defined.
      -->
<env name="DTT_HTML_OUTPUT_DIRECTORY" value="sites/simpletest/browser_output"/>
</php>

```

Screenshot using ExistingSiteSelenium2DriverTest

From <https://gitlab.com/weitzman/drupal-test-trait#debugging-tests>:

- To take a screenshot of the current page under `ExistingSiteSelenium2DriverTest`, use `\weitzman\DrupalTestTrait\ScreenShotTrait::captureScreenshot`. Be careful when using this to debug tests that are "randomly" failing. Most likely, these tests are failing due to missing `[waitForElementVisible]{.underline}` checks, as the act of taking a screenshot gives the browser additional time to finish rendering the page.

This trait doesn't need to be installed separately, it is included with DTT. Just add a use statement to your class e.g.

```

class Pub1Test extends ExistingSiteSelenium2DriverTestBase {
  use \weitzman\DrupalTestTrait\ScreenShotTrait;

  private int $teaPublisherUid = 5071;
  private int $teaPublisherNid = 61821;

  public function testSetup() {
    $this->markTestIncomplete('later');
    ...
  }
}

```

Calling it in code:

```
// Load the publisher correlation detail page.
do {
    $url = Url::fromRoute('too_teks_publisher.correlation_detail', [
        'node1' => $this->tooPublisherNid,
        'node2' => $this->testProgramOneNid,
        'node3' => $expectation_nid,
        'node4' => $correlation_nid,
    ])
};

$page_source = $this->drupalGet($url);
// $this->capturePageContent();
$this->captureScreenshot();
```

The output appears as png files like:



Curiously, this only captures the visible part of the page - I notice parts of it were clipped.

Location of the files is specified in the phpunit.xml in the <php> section as

```
<env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />
```

That whole <php> section looks like:

```

<php>
<env name="DTT_BASE_URL" value="http://tea.ddev.site"/>
<env name="DTT_API_URL" value="http://chrome:9222"/>
<env name="DTT_MINK_DRIVER_ARGS" value='["chrome", {"browserName": "chrome", "chromeOptions": {"args": ["--disable-gpu", "--headless", "--no-sandbox"]}}, "http://chrome:9222"]' />
<env name="DTT_API_OPTIONS" value='{"socketTimeout": 360, "domWaitTimeout": 3600000}' />
<!-- Example BROWSERTEST_OUTPUT_DIRECTORY value: /tmp
      Specify a temporary directory for storing debug images and html documents.
      These artifacts get copied to /sites/simpletest/browser_output by BrowserTestBase. -->
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
<!-- To disable deprecation testing completely uncomment the next line. -->
<env name="SYMFONY_DEPRECATED_HELPER" value="disabled"/>
<!-- Specify the default directory screenshots should be placed. -->
<!-- <env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />-->
<!-- Specify the default directory page captures should be placed.
      When using the \Drupal\Tests\Listeners\HtmlOutputPrinter printerClass this will default to
      /sites/simpletest/browser_output. If using another printer such as teamcity this must be defined.
      -->
<env name="DTT_HTML_OUTPUT_DIRECTORY" value="sites/simpletest/browser_output"/>
</php>

```

Writing DTT Tests

Test locations

Tests that require no Ajax or Javascript are put in the ExistingSite directory. These will run quite quickly especially if you run them on the host (instead of in the DDEV/Docker containers). These tests are derived from ExistingSiteBase.

Putting tests in the ExistingSiteJavascript directory (and deriving them from ExistingSiteSelenium2DriverTestBase) will cause the test to be run against the Chromedriver which can handle Javascript and Ajax.

Generate DTT tests with drush

Use the latest drush (11 at this time) to generate both types of tests using:

```
$ drush generate test:existing
```

And

```
$ drush generate test:existing-js
```

More at <https://www.drush.org/latest/generators/>

Example tests

ExampleTest.php creating user term, article

From <https://gitlab.com/weitzman/drupal-test-trait/-/blob/2.x/tests/ExampleTest.php> this example creates a user, a taxonomy term, an article, retrieves it, checks for a return code of 200, logs in as the user and retrieves the node edit page.

```

<?php

namespace Drupal\Tests\tea_teks_voting\ExistingSite;

use Drupal\taxonomy\Entity\Vocabulary;
use weitzman\DrupalTestTraits\ExistingSiteBase;

/**
 * A model test case using traits from Drupal Test Traits.
 */
class ExampleTest extends ExistingSiteBase {

  protected function setUp(): void {
    parent::setUp();

    // Cause tests to fail if an error is sent to Drupal logs.
    $this->failOnLoggedErrors();
  }

  /**
   * An example test method; note that Drupal API's and Mink are available.
   *
   * @throws \Drupal\Core\Entity\EntityStorageException
   * @throws \Drupal\Core\Entity\EntityMalformedException
   * @throws \Behat\Mink\Exception\ExpectationException
   */
  public function testLlama() {
    // Creates a user. Will be automatically cleaned up at the end of the test.
    $author = $this->createUser([], null, true);

    // Create a taxonomy term. Will be automatically cleaned up at the end of the test.
    $vocab = Vocabulary::load('tags');
    $term = $this->createTerm($vocab);

    // Create a "Llama" article. Will be automatically cleaned up at end of test.
    $node = $this->createNode([
      'title' => 'Llama',
      'type' => 'article',
      'field_tags' => [
        'target_id' => $term->id(),
      ],
      'uid' => $author->id(),
    ]);
    $node->setPublished()->save();
    $this->assertEquals($author->id(), $node->getOwnerId());

    // We can browse pages.
    $this->drupalGet($node->toUrl());
    $this->assertSession()->statusCodeEquals(200);

    // We can login and browse admin pages.
    $this->drupalLogin($author);
    $this->drupalGet($node->toUrl('edit-form'));
  }
}

```

VotingPageTest

This test runs code in a project and at various times asserts that various things are true or equal to expected values. It also does some setup

including logging in as a voter. More about that below.

```
class VotingPageTest extends ExistingSiteBase {

protected int $testProgramOneNid = 852061;
protected int $publisher_user_id = 3501; // Danny Test Lufkin.
protected int $correlationNid = 852086;
protected int $expectationNid = 852076;
protected int $teaPublisherNid = 61821;
protected ?int $adminUserId = NULL;

public function testVoter1Vote() {
//  $this->markTestIncomplete('later');

$this->setupTestProgram1ForVotingRound0();

$this->loginVoter1();
$program_nid = $this->testProgramOneNid;
$this->program = new Program($program_nid);
$this->persona = new Persona($this->voter1UserId);

$citation_nid = 858676;
$citation_nid = 858741;
$citation_node = Node::load($citation_nid);
$expectation_nid = $citation_node->get('field_tks_pub_expectation')->target_id;
$correlation_nid = $citation_node->get('field_tks_pub_correlation')->target_id;

$options = [
'program_nid' => $program_nid,
'expectation_nid' => $expectation_nid,
'correlation_nid' => $correlation_nid,
'citation_nid' => $citation_nid,
'rejection_reason' => 'auto-test rejection reason',
];
}

$this->votingProcessor = \Drupal::service('tea_teks_srp.vote_processor');
$correlation_node = Node::load($correlation_nid);
$valid_voting_path = $this->votingProcessor->loadVotingPath($this->program->getProgramNode(), 'all');
$this->votingProcessor->setVotingRequirements($correlation_node);
self::assertTrue($valid_voting_path);

$this->votingProcessor->voteOnCitation($options, 'accepted');
$voting_record_node = $this->votingProcessor->loadVotingRecord($citation_nid, $this->voter1UserId);
$vote = $voting_record_node->get('field_vote')->value;
self::assertEquals('accepted', $vote);

$this->votingProcessor->voteOnCitation($options, 'rejected');
$voting_record_node = $this->votingProcessor->loadVotingRecord($citation_nid, $this->voter1UserId);
$vote = $voting_record_node->get('field_vote')->value;
self::assertEquals('rejected', $vote);

$this->votingProcessor->cancelVoteOnCitation($citation_nid, $correlation_nid);
$voting_record_node = $this->votingProcessor->loadVotingRecord($citation_nid, $this->voter1UserId);
self::assertEmpty($voting_record_node);
}

}
```

Here is the setupTestProgram1ForVoting

This checks some data in nodes, logs in as an admin user and fills out a form that executes a batch api process. I wasn't able to make a batch api process run directly from a test. Not sure why. Moshe Weitzman says it should work but that the batch API is ancient.

```
protected function setupTestProgram1ForVotingRound0() {
  // $this->resetProgram1Voting();

  $program_node = Node::load($this->testProgramOneNid);
  $teks_program_status = $program_node->get('field_tks_program_status')->value;
  if ($teks_program_status == 'publisher_complete') {
    $this->loginAdminUser();
    $page_source = $this->drupalGet(Url::fromRoute('tea_teks_publisher.change_input_collection_status', ['node' => 852071,]));
    // To check if the form displayed correctly you can look for something in the $page_source and check the return code.
    $this->assertSession()->statusCodeEquals(200);

    $this->submitForm([
      'program_status' => 'ready_for_release',
    ], 'Change Status');

    // Confirm that the voting requirements were written to a correlation.
    $correlation_node = Node::load(852081);
    $voting_requirements_json = $correlation_node->get('field_voting_requirements_json')->value;
    self::assertNotNull($voting_requirements_json);
  }
  // Set team.
  $this->setupVotingTeamA();
  $this->setupTestProgram1TeamAForRound0();
}
```

Other supporting functions

```
protected function setupVoter1() {
  $query = \Drupal::entityQuery('user')
    ->condition('name', 'Voter 1 Test');
  $uids = $query->execute();
  if (empty($uids)) {
    $this->setupVotingTeamA();
    $query = \Drupal::entityQuery('user')
      ->condition('name', 'Voter 1 Test');
    $uids = $query->execute();
  }
  $uids = array_values($uids);
  $this->voter1UserId = $uids[0];
}
```

and

```
protected function loginVoter1() {
  $this->setupVoter1();
  $user = User::load($this->voter1UserId);
  $user->passRaw = 'password';
  $this->drupalLogin($user);
}
```

ExampleLoginTest.php

From <https://gitlab.com/weitzman/logintrait/-/blob/master/src/ExampleLoginTest.php> This example shows how to use the Login trait.

```
<?php

// Use your module's testing namespace such as the one below.
namespace Drupal\Tests\moduleName\ExistingSite;

use weitzman\Drupal\Test\Traits\ExistingSiteBase;
use weitzman\LoginTrait>LoginTrait;

/**
 * Login and Logout via user reset URL instead of forms. Useful when TFA/SAML are enabled.
 */
class ExampleLoginTest extends ExistingSiteBase {
    use LoginTrait;

    /**
     * Login and logout via password reset URL.
     */
    public function testLoginLogout() {
        // Creates a user. Will be automatically cleaned up at the end of the test.
        $user = $this->createUser();
        $this->drupalLogin($user);
        $this->drupalGet('user');
        $user2 = $this->createUser();
        $this->drupalLogin($user2);
    }
}
```

Selenium2DriverTest

This is an example from the Weitzman repo using AJAX

```

<?php

// Use your module's testing namespace such as the one below.
//namespace Drupal\Tests\tea_teks_voting\ExampleSelenium2DriverTest;
namespace Drupal\Tests\tea_teks_voting\ExistingSiteJavascript;

use Drupal\node\Entity\Node;
use Drupal\taxonomy\Entity\Vocabulary;
use Drupal\user\Entity\User;
use weitzman\DrupalTestTraits\ExistingSiteSelenium2DriverTestBase;

/**
 * A WebDriver test suitable for testing Ajax and client-side interactions.
 */
class ExampleSelenium2DriverTest extends ExistingSiteSelenium2DriverTestBase
{
    public function testContentCreation()
    {
        // Create a taxonomy term. Will be automatically cleaned up at the end of the test.
        $web_assert = $this->assertSession();
        $vocab = Vocabulary::load('tags');
        $this->createTerm($vocab, ['name' => 'Term 1']);
        $this->createTerm($vocab, ['name' => 'Term 2']);
        $admin = User::load(1);
        $admin->passRaw = 'password';
        $this->drupalLogin($admin);

        // @codingStandardsIgnoreStart
        // These lines are left here as examples of how to debug requests.
        // \weitzman\DrupalTestTraits\ScreenShotTrait::captureScreenshot();
        // $this->capturePageContent();
        // @codingStandardsIgnoreStop

        // Test autocomplete on article creation.
        $this->drupalGet('/node/add/article');
        $page = $this->getCurrentPage();
        $page->fillField('title[0][value]', 'Article Title');
        $tags = $page->findField('field_tags[target_id]');
        $tags->setValue('Ter');
        $tags->keyDown('m');
        $result = $web_assert->waitForElementVisible('css', '.ui-autocomplete li');
        $this->assertNotNull($result);
        // Click the autocomplete option
        $result->click();
        // Verify that correct the input is selected.
        $this->assertStringContainsString('Term 1', $tags->getValue());
        $submit_button = $page->findButton('Save');
        $submit_button->press();
        // Verify the URL and get the nid.
        $this->assertTrue((bool) preg_match('/.+node\/(?P<nid>\d+)/', $this->getUrl(), $matches));
        $node = Node::load($matches['nid']);
        $this->markEntityForCleanup($node);
        // Verify the text on the page.
        $web_assert->pageTextContains('Article Title');
        $web_assert->pageTextContains('Term 1');
    }
}

```

Login to your site

There are some contributed packages with useful features. The [login trait repo](#) adds some useful functionality.

Install via Composer with:

```
$ composer require weitzman/logintrait
```

Create a new user and login as that user:

From <https://gitlab.com/weitzman/logintrait/-/blob/master/src/ExampleLoginTest.php>

```
<?php

// Use your module's testing namespace such as the one below.
namespace Drupal\Tests\moduleName\ExistingSite;

use weitzman\DrupalTestTraits\ExistingSiteBase;
use weitzman>LoginTrait>LoginTrait;

/**
 * Login and Logout via user reset URL instead of forms. Useful when TFA/SAML are enabled.
 */
class ExampleLoginTest extends ExistingSiteBase {
  use LoginTrait;

  /**
   * Login and logout via password reset URL.
   */
  public function testLoginLogout() {
    // Creates a user. Will be automatically cleaned up at the end of the test.
    $user = $this->createUser();
    $this->drupalLogin($user);
    $this->drupalGet('user');
    $user2 = $this->createUser();
    $this->drupalLogin($user2);
  }
}
```

Create an admin user

This will create a user named Fred Bloggs who is in a new randomly named group. The user and the group will be deleted when the test run finishes.

```
// Creates a user. Will be automatically cleaned up at the end of the test.
$user = $this->createUser([], 'Fred Bloggs', TRUE);
$this->drupalLogin($user);
$this->drupalGet('user');
```

Login as an existing user

Be sure to set that user's password to "password" (in the Drupal U/I or in code) in order to make this work.

```
$voter1_user_id = 5284;
$voter1 = User::load($voter1_user_id);
$voter1->passRaw = 'password';
$this->drupalLogin($voter1);
```

Add users using Drupal API

```

$voter1 = User::create([
  'name' => $this->voter1['name'],
  'field_firstname' => $this->voter1['field_firstname'],
  'field_lastname' => $this->voter1['field_lastname'],
  'field_srp_voter_role' => $this->voter1['field_srp_voter_role'],
  'field_phone' => $this->voter1['field_phone'],
  'field_title' => $this->voter1['field_title'],
]);
$voter1->addRole('srp_voter');
$voter1->setEmail('voter1@mightycitizen.com');
$voter1->setPassword('password');
$voter1->activate();
$voter1->save();
$Voter1_uid = $voter1->id();

```

Fill out a form

Here is the code from docroot/core/tests/Drupal/Tests/UiHelperTrait.php to fill out the login form:

```

$this->drupalGet(Url::fromRoute('user.login'));
$this->submitForm([
  'name' => $account->getAccountName(),
  'pass' => $account->passRaw,
], 'Log in');

```

Here is another example:

```

// Load the form.
$url = Url::fromRoute('tea_teks_admin.sanity_checker', [
  'program' => $this->testProgramOneNid,
]
);
// Confirm that it loaded without errors.
$this->assertSession()->statusCodeEquals(200);
// Check the destructive checkbox and click the 'verify vote counts' button.
$this->submitForm(['destructive' => 1], 'Verify Vote Counts');

```

In the above example the code in render array for the form that builds the destructive checkbox and the submit button looks like this:

```

$form['sanity_fieldset']['destructive'] = [
  '#type' => 'checkbox',
  '#title' => t('Check this box to permanently update statuses.'),
  '#description' => t('Recalculate all votes and statuses for current vote number. Leave unchecked for testing.'),
];
$form['sanity_fieldset']['actions'] = [
  '#type' => 'actions',
];
$form['sanity_fieldset']['actions']['submit'] = [
  '#type' => 'submit',
  '#value' => $this->t('Verify Vote Counts'),
];

```

Parameter gotcha

And my effort to fill out a form with a dropdown. This route required a node id to be passed as a parameter to the form – hence the [node'=> 852071] and my submit button is called “Change Status”

```
$this->drupalGet(Url::fromRoute('tea_teks_publisher.change_input_collection_status', ['node' => 852071,]));
$this->submitForm([
  'program_status' => 'ready_for_release',
], 'Change Status');
```

Note. When you define a form in Drupal, it permits you to use a different variable name in the routing file versus the parameter in the buildForm() function. E.g. Here the parameter is called "program":

```
tea_teks_srp.reset_program_votes:
path: '/teks/admin/srp/program/{program}/resetvotes'
defaults:
_form: '\Drupal\tea_teks_srp\Form\SrpResetProgramVotesForm'
_title: 'Reset Program Votes'
requirements:
_permission: 'manage teks srp process'
options:
parameters:
program:
type: entity:node
no_cache: TRUE'
```

In the form, the parameter can be something different. i.e. the \$node parameter here represents the program parameter above. If you change them to match i.e. change the parameter in the buildForm function below, it should work fine.

```
public function buildForm(array $form, FormStateInterface $form_state, EntityInterface $node = NULL) {
$form['#theme'] = 'tea_teks_srp__reset_votes';
if ( ($node->id() && $node->bundle() == 'teks_pub_program') ) {
$request = \Drupal::request();
$referer = $request->headers->get('referer');
$base_url = Request::createFromGlobals()->getSchemeAndHttpHost();
$alias = substr($referer, strlen($base_url));
$form_state->set('referrer_alias', $alias);
$current_user = \Drupal::currentUser();
if ($current_user->hasPermission('manage teks srp process')) {
$form['actions'] = [
'#type' => 'actions',
];
$form['actions']['submit'] = [
'#type' => 'submit',
'#value' => $this
->t('TESTING ONLY: Reset Program Votes/Data'),
];
$form_state->set('program_id', $node->id());
$form_state->set('program_title', $node->title->value);
}
}
return $form;
}
```

So if you try in the test to execute this form and pass it a parameter called "program" it will fail to load the form. You will see errors like:

There was 1 error:

1) Drupal\Tests\tea_teks_voting\ExistingSite\Vote1::testSetup

Behat\Mink\Exception\ElementNotFoundException: Button with id|name|label|value "TESTING ONLY: Reset Program Votes/Data" not found.

Note. This code will do the same thing if you put it in the ExistingSite or the ExistingSiteJavascript directory however, putting it in the ExistingSiteJavascript directory (and deriving the test from ExistingSiteSelenium2DriverTestCase) will cause the test to be run against the

Chromedriver which can handle Javascript and Ajax.

Data Provider

Tests can be repeated with varying values by providing a data provider function. The data provider just returns an array of values and the function below has annotation indicating to PHPUnit to rerun the test once for each value in the data provider.

```
public function providerForTest1(): array {
    return [
        [858641, 'accepted'],
        [858651, 'accepted'],
        [858661, 'accepted'],
        [858676, 'accepted'],
        [858721, 'accepted'],
        [858726, 'accepted'],
        [858736, 'accepted'],
        [858741, 'accepted'],
    ];
}

/**
 * @dataProvider ProviderForTest1
 */
public function test1(int $citation_nid, string $vote) {
    static $voter1_uid = 0;
    $this->setupTestProgram1ForVotingRound0();
    $this->loginVoter1();
    $this->program = new Program($this->testProgramOneNid);
    $this->persona = new Persona($voter1_uid);
    $this->votingProcessor = \Drupal::service('tea_teks_srp.vote_processor');
    $valid_voting_path = $this->votingProcessor->loadVotingPath($this->program->getProgramNode(), 'all');
    self::assertTrue($valid_voting_path);

    $citation_node = Node::load($citation_nid);
    $correlation_nid = $citation_node->get('field_tks_pub_correlation')->target_id;
    $expectation_nid = $citation_node->get('field_tks_pub_expectation')->target_id;
    $correlation_node = Node::load($correlation_nid);
    $this->votingProcessor->setVotingRequirements($correlation_node);
    $voting_options = [
        'program_nid' => $this->testProgramOneNid,
        'expectation_nid' => $expectation_nid,
        'correlation_nid' => $correlation_nid,
        'citation_nid' => $citation_nid,
        'rejection_reason' => 'auto-test rejection reason',
    ];
    $this->votingProcessor->voteOnCitation($voting_options, 'accepted');

    $voting_record_node = $this->votingProcessor->loadVotingRecord($citation_nid, $this->voter1UserId);
    $vote_value_from_node = $voting_record_node->get('field_vote')->value;
    self::assertEquals($vote, $vote_value_from_node);
}
```

Fill a queue and run a trait

From Moshe Weitzman 9-27-22

I have seen tests that fill a queue and then run the queue with <https://github.com/drupaltest/queue-runner-trait/>

//@TODO: Explore this

Mink

Checking page return code

This only works for non-Selenium/Chromedriver type test:

```
$session = $this->getSession();
$status_code = $session->getStatusCode();
print "\n Current Status code: $status_code";
```

In the ExampleTest.php there was this example:

```
// We can browse pages.
$this->drupalGet($node->toUrl());
$this->assertSession()->statusCodeEquals(200);
```

Grab the text from the page

You can do some interesting things when running Selenium type tests. Here we can grab the text and search in it for a particular string.

This will get you the text that is visible on the page. It is unformatted and is one long string.

```
$session = $this->getSession();
$page = $session->getPage();
$page_text = $page->getText();
```

Current URL

```
$url_string = $this->getSession()->getCurrentUrl();
print "\n Current URL: $url_string";
```

Load and parse a CSV file

I found it useful for tests to be able to load a CSV file to drive a test by inputting repeatable data over and over. This is similar to using a data provider function.

```

private function readCsv2(): array {
    $file = getcwd() . '/modules/custom/tea_teks/modules/tea_teks_publisher/tests/ExistingSiteJavascript/test2.csv';
    $csv = array_map('str_getcsv', file($file));
    array_walk($csv, function(&$a) use ($csv) {
        $a = array_combine($csv[0], $a);
    });
    array_shift($csv); # remove column header

    /*
     * The above code produces arrays of values for the CSV file:
     */
    [0] => [
        [Num] => 1
        [Program] => 852061
        [Expectation] => 852076
        [Correlation] => 852081
        [SKIP] => N
        [Romanette] => i
        [KSS-SE] => met
        [xofy] => 1 1 1 1
        [Citations] => SN, SA
        [BrkStatus] => complete
        [ExpecStatus] => unmet
    ]
    /*
     * Use this code to display it on screen:
     */
    echo '<pre>';
    print_r($csv);
    echo '</pre>';
    *
    */
    return $csv;
}

```

Here is the CSV file:

```

**Num**, **Program**, **Expectation**, **Correlation**, **SKIP**, **Romanette**, **KSS-SE**, **xofy**, **Citations**, **BrkStatus**, **ExpecStatus\

1**, **852061**, **852076**, **852081**, **N**, **i**, **1.A**, **"1, 1, 1, 1", "SN, SA", met, unmet
1\**\*, **"SN, SA"\**, **met\**, **unmet\

2**, **852061**, **852076**, **852086**, **N**, **ii**, **1.A**, **"1, 1, 1", "SN, SA, TN, TA", met, unmet
1\**\*, **"SN, SA, TN, TA"\**, **met\**, **unmet\

3**, **852061**, **852076**, **852091**, **N**, **iii**, **1.A**, **"0, 1, 0", "SA", unmet, unmet
1\**\*, **"SA"\**, **unmet\**, **unmet\

4**, **852061**, **852076**, **852096**, **N**, **iv**, **1.A**, **"0, 1, 0", "SA", unmet, unmet
1\**\*, **"SA"\**, **unmet\**, **unmet\

5**, **852061**, **852161**, **852166**, **N**, **ij**, **1.B**, **"0, 0, 1", "SA", unmet, unmet
1\**\*, **"SA, SA, SA, SA, TN, TA"\**, **unmet\**, **unmet\

6**, **852061**, **852161**, **852171**, **N**, **ji**, **1.B**, **"0, 1, 1", "TN, TN, TN, TN, TA", unmet, unmet
1\**\*, **"TN, TN, TN, TN, TA"\**, **unmet\**, **unmet\

8**, **852061**, **852101**, **852111**, **N**, **jj**, **2.A**, **"SN, SA, SA, TN, TA", unmet, unmet
SA, TN, TN, TA, TA\**\*, **met\**, **unmet\

9**, **852061**, **852116**, **852121**, **N**, **ij**, **2.B**, **"SN, SA, TN, TA", unmet, unmet
TA\**\*, **unmet\**, **unmet\

10**, **852061**, **852116**, **852126**, **N**, **ji**, **2.B**, **"SN, SA, TN, TA", unmet, unmet
SA\**\*, **unmet\**, **unmet\

11**, **852061**, **852131**, **852136**, **N**, **i**, **3.A**, **"SN, SN, SN, SA, SA, SA", met, met
SN, SN, SA, SA, SA, SA\**\*, **met\**, **met\

12**, **852061**, **852131**, **852141**, **N**, **ii**, **3.A**, **"TN, TN, TN, TN, TA", met, met
13**, **852061**, **852146**, **852151**, **N**, **i**, **3.B**, **"TA", met, met
14**, **852061**, **852146**, **852156**, **N**, **ii**, **3.B**, **"TN, TA", met, met
TA\**\*, **met\**, **met\**

```

Picture of CSV file with color formatting:

Num	Program	Expectation	Correlation	SKIP	Romanette	KSS-SE	xofy	Citations	BrkStatus	ExpecStatus
1, 852061, 852076, 852081, N, i, 1.A, "1, 1, 1, 1", "SN, SA", met, unmet										
2, 852061, 852076, 852086, N, ii, 1.A, "1, 1, 1", "SN, SA, TN, TA", met, unmet										
3, 852061, 852076, 852091, N, iii, 1.A, "0, 1, 0", "SA", unmet, unmet										
4, 852061, 852076, 852096, N, iv, 1.A, "0, 1, 0", "SA", unmet, unmet										
5, 852061, 852161, 852166, N, i, 1.B, "0, 0, 1, 1", "TN, TA", unmet, unmet										
6, 852061, 852161, 852171, N, ii, 1.B, "0, 1, 1", "SA, SA, SA, TN, TA", unmet, unmet										
7, 852061, 852101, 852106, N, i, 2.A,, "TN, TN, TN, TN, TA", unmet, unmet										
8, 852061, 852101, 852111, N, ii, 2.A,, "SN, SA, SA, TN, TN, TA, TA", met, unmet										
9, 852061, 852116, 852121, N, i, 2.B,, "SN, TA", unmet, unmet										
10, 852061, 852116, 852126, N, ii, 2.B,, "SN, SA", unmet, unmet										
11, 852061, 852131, 852136, N, i, 3.A,, "SN, SN, SN, SA, SA, SA", met, met										
12, 852061, 852131, 852141, N, ii, 3.A,, TN, met, met										
13, 852061, 852146, 852151, N, i, 3.B,, TA, met, met										
14, 852061, 852146, 852156, N, ii, 3.B,, "TN, TA", met, met										

Adding DTT to an existing site

Install DTT and dev requirements with:

Follow these steps to quickly get DTT running on your project.

```
$ composer require weitzman/drupal-test-trait --dev
```

```
$ composer require drupal/core-dev --dev --update-with-all-dependencies
```

Setup `phpunit.xml` in the root of the project (not docroot or web). There will usually be a `phpunit.xml.dist` file there. Use that file and add your tweaks to it using <https://gitlab.com/weitzman/drupal-test-trait/-/blob/master/docs/phpunit.xml> as the basis.

Create `phpunit.xml` file

My `phpunit.xml.dist`:

```
<?xml version="1.0" encoding="UTF-8"?>

<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
    backupGlobals="false"
    colors="true"
    bootstrap="vendor/autoload.php"
    verbose="true"
    >

<testsuites>
    <testsuite name="drupal-composer-project tests">
        <directory>./test/</directory>
    </testsuite>
</testsuites>
</phpunit>
```

My `phpunit.xml` with edits for site `tea3.ddev.site`. replace `tea3` with the sitename for your ddev site in the `<env name="DTT_BASE_URL" value="http://tea3.ddev.site"/>`.

```

<?xml version="1.0" encoding="UTF-8"?>

<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
    backupGlobals="false"
    colors="true"
    bootstrap="scripts/bootstrap-fast.php"
    verbose="true"
    >
</php>
<env name="DTT_BASE_URL" value="http://tea3.ddev.site"/>
<env name="DTT_API_URL" value="http://chrome:9222"/>
<env name="DTT_MINK_DRIVER_ARGS" value='["chrome", {"browserName": "chrome", "chromeOptions": {"args": ["--disable-gpu", "--headless", "--no-sandbox"]}}, "http://chrom'>
<env name="DTT_API_OPTIONS" value='{"socketTimeout": 360, "domWaitTimeout": 3600000}' />
<!-- Example BROWSERTEST_OUTPUT_DIRECTORY value: /tmp
    Specify a temporary directory for storing debug images and html documents.
    These artifacts get copied to /sites/simpletest/browser_output by BrowserTestBase. -->
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
<!-- To disable deprecation testing completely uncomment the next line. -->
<env name="SYMFONY_DEPRECATIONS_HELPER" value="disabled"/>
<!-- Specify the default directory screenshots should be placed. -->
<!--<env name="DTT_SCREENSHOT_REPORT_DIRECTORY" value="" />-->
<!-- Specify the default directory page captures should be placed.
    When using the \Drupal\Tests\Listeners\HtmlOutputPrinter printerClass this will default to
    /sites/simpletest/browser_output. If using another printer such as teamcity this must be defined.
    -->
<!--<env name="DTT_HTML_OUTPUT_DIRECTORY" value="" />-->
</php>
<testsuites>
<testsuite name="drupal-composer-project tests">
    <directory>./test/</directory>
</testsuite>
<testsuite name="unit">
    <directory>./web/modules/custom/*/tests/src/Unit</directory>
    <!--<directory>./web/profiles/custom/*/tests/src/Unit</directory>-->
</testsuite>
<testsuite name="kernel">
    <directory>./web/modules/custom/*/tests/src/Kernel</directory>
    <!--<directory>./web/profiles/custom/*/tests/src/Kernel</directory>-->
</testsuite>
<testsuite name="existing-site">
    <!-- Assumes tests are namespaced as \Drupal\Tests\custom_foo\ExistingSite. -->
    <directory>./web/modules/custom/*/tests/src/ExistingSite</directory>
    <!--<directory>./web/profiles/custom/*/tests/src/ExistingSite</directory>-->
</testsuite>
<testsuite name="existing-site-javascript">
    <!-- Assumes tests are namespaced as \Drupal\Tests\custom_foo\ExistingSiteJavascript. -->
    <directory>./web/modules/custom/*/tests/src/ExistingSiteJavascript</directory>
    <!--<directory>./web/profiles/custom/*/tests/src/ExistingSiteJavascript</directory>-->
</testsuite>
</testsuites>
</phpunit>

```

Create bootstrap-fast.php

Create `/scripts/bootstrap-fast.php` with the following contents

```

<?php
/**
 * @file
 *
 * A bootstrap file for `phpunit` test runner.
 *
 * This bootstrap file from DTT is fast and customizable.
 *
 * If you get 'class not found' errors while running tests, you should copy this
 * file to a location inside your code-base --such as `/scripts`. Then add the
 * missing namespaces to the bottom of the copied field. Specify your custom
 * `bootstrap-fast.php` file as the bootstrap in `phpunit.xml`.
 *
 * Alternatively, use the `bootstrap.php` file, in this same directory, which is
 * slower but registers all the namespaces that Drupal tests expect.
*/
use Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter;
use weitzmanDrupalTestTraits\AddPsr4;

list($finder, $class_loader) = AddPsr4::add();
$root = $finder->getDrupalRoot();

// So that test cases may be simultaneously compatible with multiple major versions of PHPUnit.
$class_loader->addPsr4('Drupal\TestTools\\', "$root/core/tests");
if (class_exists('Drupal\TestTools\PhpUnitCompatibility\PhpUnit8\ClassWriter')) {
  ClassWriter::mutateTestBase($class_loader);
}

// Register more namespaces, as needed.
# $class_loader->addPsr4('Drupal\Tests\my_module\\', "$root/modules/custom/my_module/tests/src");

```

Add .phpunit.result.cache file to .gitignore

To stop result cache getting checked into the repo, add the `.phpunit.result.cache` to the `.gitignore` file.

You could also change this file location by editing `phpunit.xml`:

```

<phpunit
  ...
  cacheResultFile=".temp/fs_cache/.phpunit.result.cache"
>
```

Or completely disable it by:

```

<phpunit
  ...
  cacheResult ="false"
>
```

Remove DTT and core-dev

For production deployment, you can remove DTT and core-dev with:

```

$ composer remove drupal/core-dev --dev

$ composer remove weitzman/drupal-test-trait --dev
```

Alternatively, just run

```
$ composer update --no-dev
```

Hide deprecation notices

To hide deprecation notices when running test on the host, update your php.ini (run `php --ini` to find the `php.ini` file) and change the `error_reporting` line from:

```
error_reporting = E_ALL
```

to:

```
error_reporting = E_ALL & ~E_DEPRECATED
```

Now tests should look like this:

```
$ vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/VotingPageTest.php

PHPUnit 9.5.24 #StandWithUkraine

Runtime: PHP 8.1.9

Configuration: /Users/selwyn/Sites/tea/phpunit.xml

. 1 / 1 (100%)

Time: 00:00.830, Memory: 46.50 MB

OK (1 test, 3 assertions)
```

Troubleshooting DTT Tests

Which test?

For tests that involve the Drupal API, if a test fails, you might see output like this:

```
1) Drupal\Tests\tea_teks_requirements\ExistingSite\Requirements1Test::testEmptyRequirements
ArgumentCountError: Too few arguments to function Drupal\Core\Entity\EntityBase::load(), 0 passed in /var/www/html/web/modules/custom/tea_teks_requirements/tests/src/ExistingSite/Requirements1Test.php
[1] [2]
```

This is indicating the first test by: "1)". If this were the second test in the file, it would show "2)". The error is that too few arguments were passed to `EntityBase::load()` – in my case, I was passing null to a `Node::load()` function.

PolyfillAssertTrait not found

After installing DTT you see errors when you try to run the tests like this:

```
selwyn@tea3-web:/var/www/html$ ./vendor/bin/phpunit --bootstrap=./vendor/weitzman/drupal-test-trait/src/bootstrap-fast.php
./docroot/modules/custom/tea_teks/modules/tea_teks_requirements/tests/src/ExistingSite/RequirementsCreationTest.php

PHP Fatal error: Trait "Symfony\Bridge\PhpUnit\Legacy\PolyfillAssertTrait" not found in /var/www/html/docroot/sites/simpletest/Assert.php
on line 91

Fatal error: Trait "Symfony\Bridge\PhpUnit\Legacy\PolyfillAssertTrait" not found in /var/www/html/docroot/sites/simpletest/Assert.php
on line 91
```

You will need to install the dev requirements with:

```
$ composer require drupal/core-dev --dev
--update-with-all-dependencies
```

Class not found errors

If you see something like this on your brand new class you created:

```
1) Drupal\Tests\tea_teks_voting\ExistingSite\ProgramTest::testIsVotingPermitted
Error: Class "Drupal\tea_teks_voting\Program" not found
```

This is a real forehead slapper! Be sure to enable your custom module under Drupal's extend menu option.

Also

When running the tests, if they start throwing "class not found" errors, this may indicate some outdated code in your codebase.

```
$ vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/Tests/src/ExistingSite/RequirementsCreationTest.php
```

```
PHP Fatal error: Uncaught Error: Class "PHPUnit\TextUI\Command" not found in /var/www/html/vendor/phpunit/phpunit/phpunit:98
```

Stack trace:

```
#0 /var/www/html/vendor/bin/phpunit(123): include()
#1 {main}
thrown in /var/www/html/vendor/phpunit/phpunit on line 98
Fatal error: Uncaught Error: Class "PHPUnit\TextUI\Command" not found in /var/www/html/vendor/phpunit/phpunit/phpunit:98
Stack trace:
#0 /var/www/html/vendor/bin/phpunit(123): include()
#1 {main}
thrown in /var/www/html/vendor/phpunit/phpunit/phpunit on line 98
```

The fix in this case was a composer update.

Also If you use another test as a starting point (ie. Copy the file) and forget to change the class name, that would cause a similar error:

```
$ vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/PersonaTest.php
```

```
Class 'PersonaTest' could not be found in '/Users/selwyn/Sites/tea/docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/PersonaTest.php'.
```

In this example, the filename was PersonaTest.php but the class name is accidentally called TeamTest so the interpreter could not find a PersonaTest. Oops. Here is the errant PersonaTest.php file:

```
<?php

namespace Drupal\Tests\tea_teks_voting\ExistingSite;

use Drupal\node\Entity\Node;
use Drupal\tea_teks_voting\Persona;
use weitzman\Drupal\Test\Traits\ExistingSiteBase;

class TeamTest extends ExistingSiteBase {

  public function testPersonaEmptyLoading() {
    $p = new Persona();
    $user_id = $p->getUserId();
    self::assertEquals(0, $user_id);
  }

}

var_dump, echo, print
```

For quick variable dumps, it is quite valid to use var_dump in your tests. Here is a test with a var_dump() call.

You can also print or echo variables e.g.

```
// \n will put this output on a new line.
```

```
$temp = "blah";
echo "\nResults = $temp";
print "\nResults = $temp";
```

```
<?php
```

```
namespace Drupal\Tests\tea_teks_voting\ExistingSite;
```

```
use Drupal\node\Entity\Node;
use Drupal\tea_teks_voting\Team;
use weitzmanDrupalTestTraits\ExistingSiteBase;
```

```
class TeamTest extends ExistingSiteBase {
```

```
    public function testTeamLoading() {
        // Team D - Voter 1, 2, 3
        // 5101, 5106, 5116
        $t = new Team(868296);
        $team_member_info = $t->getTeamMemberInfo(5101);
        var_dump($team_member_info);
        $this->assertSame($team_member_info['name'], 'Voter 1');
        $this->assertSame($team_member_info['fullname'], 'Voter 1 Test');
        $this->assertSame($team_member_info['mail'], 'voter1@mightycitizen.com');
        $this->assertSame($team_member_info['roles'][0], 'authenticated');
        $this->assertSame($team_member_info['roles'][1], 'srp_voter');
    }
}
```

Here is the output. Note. I removed the deprecated messages for clarity. First the command to run the test:

```
$ vendor/bin/phpunit docroot/modules/custom/tea_teks/modules/tea_teks_voting/tests/src/ExistingSite/TeamTest.php
```

and the output:

```
PHPUnit 9.5.24 #StandWithUkraine

Runtime: PHP 8.1.9

Configuration: /Users/selwyn/Sites/tea/phpunit.xml

.

1 / 1 (100%)

array(11) {
    ["uid"]=> string(4) "5101"
    ["fullname"]=> string(12) "Voter 1 Test"
    ["name"]=> string(7) "Voter 1"
    ["mail"]=> string(24) "voter1@mightycitizen.com"
    ["status"]=> string(1) "1"
    ["firstname"]=> string(7) "Voter 1"
```

```

["lastname"]=> string(4) "Test"
["title"]=> NULL
["phone"]=> NULL
["roles"]=> array(2) {
[0]=> string(13) "authenticated"
[1]=> string(9) "srp_voter"
}
["voter_role"]=> string(8) "educator"
}

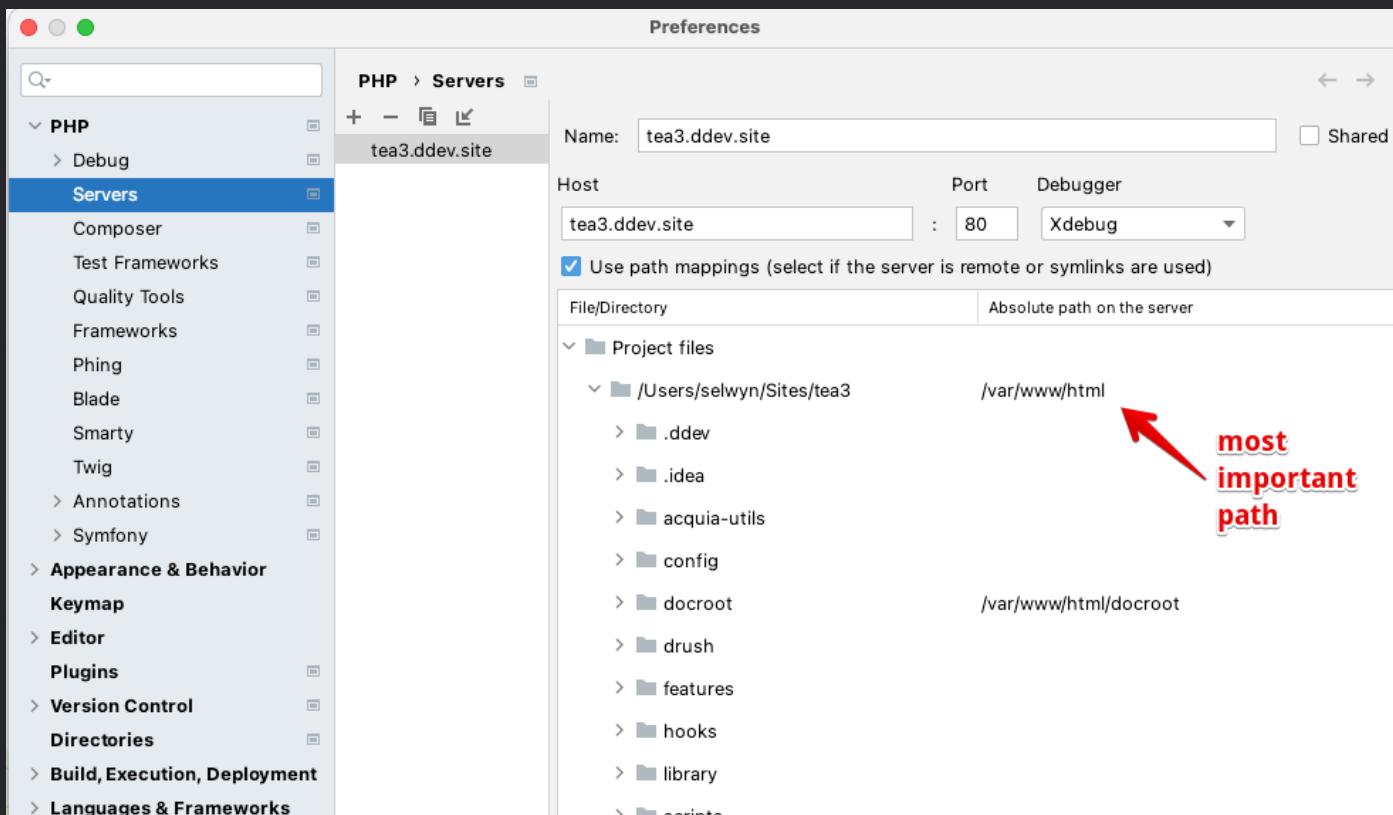
Time: 00:00.773, Memory: 44.50 MB

OK (1 test, 5 assertions)

```

Using Xdebug and PHPStorm to debug DTT scripts

It is easiest to make sure you have PHPStorm Xdebug working first, then make sure the path mappings are correct. Note. This process is almost identical to debugging drush commands.



```
$ ddev exec enable_xdebug
$ ddev ssh
```

Note. Servername has to match theservername in your phpstorm setup on a per project – see screenshot below.

Sometimes this step doesn't seem to be required. Not sure why.

```
$ export PHP_IDE_CONFIG="serverName=tea.ddev.site"
```

(the tea part above needs to match your ddev project name. e.g. drupal.ddev.site or selwyn.ddev.site etc.)

click listen in PHPStorm

click on the line number in PHPStorm to set a breakpoint

Issue the phpunit command in the vendor directory:

```
$ vendor/bin/phpunit  
docroot/modules/custom/tea_teks/modules/tea_teks_requirements/Tests/src/ExistingSite/RequirementsCreationTest.php
```

When PhpStorm pops up, specify that the vendor directory is at `/var/www/html/vendor` - note you only have to do that once and then PhpStorm will remember it.

Resources

General

- Video intro to DTT with Moshe Weitzman from DrupalCon Global Sep 2020 <https://www.youtube.com/watch?v=TExPFQ1-AA0&t=10s>
- Unit Tests in Drupal: the road to test-driven development video from DrupalCon Global 2020 by CivicActions folks: Gerardo Gonzalez and Eric Napier. This gets into the details but is hard to make out the code as it is small. <https://www.youtube.com/watch?v=UGq3G0dsAGw>
- Understanding Automated Tests in Drupal video from DrupalGov 2020 by Ridhima Abrol & Sujeet Kumar Verma. They are showing examples using MAMP. This covers unit vs kernel vs functional vs functional JS. There are fairly legible code examples. <https://www.youtube.com/watch?v=kQEMnk4boP4&list=TLPOQMTcwODlwMjKbpqkPMTkhxA&index=4>
- How to unit test your code in Drupal 8 video by Daniel Nitsche for DrupalSouth 2017 1-10-2017 (Loft??). Australian chap (very hands-on) going through live demo on PhpStorm running phpunit on Drupal 8. Using mockery. <https://www.youtube.com/watch?v=7FjjZ3OoD6Y>
- Running and debugging PHPUnit tests in PhpStorm with ddev and xdebug video by Australian Michael Strelan on 8-18-21. The audio is very soft. He walks through details of setting up PhpStorm for DDEV etc. <https://www.youtube.com/watch?v=OdoEyY8KI9w>. There is also a companion article from Michael Strelan at <https://www.previousnext.com.au/blog/running-and-debugging-phpunit-tests-phpstorm-ddev-and-xdebug> and a repo: <https://github.com/mstrelan/ddev-phpunit-demo>. It is useful to mention that since this video was created there is a [DDEV integration plugin](#) for PhpStorm which automatically configures things like path mappings, cli interpreters and phpunit configuration. This makes some of what he describes a lot easier.
- From a discord chat with Randy Fey and \@shaal. This may be useful to explore phpunit tests on tugboat when we get that all set up. Here's a good PR where we added phpunit into DrupalPod <https://github.com/shaal/DrupalPod/pull/41/files>
- Benji Fisher Drupal Testing repo from February 2022. Benji set up this repository to help with testing Drupal modules for coding standards and Drupal 10 compatibility. It is based on drupal/recommended-project with some parts borrowed from Matt Glaman's [Drupal & Nightwatch.js training](#). Main features: DDev configuration (mostly standard), upgrade Status module, custom DDev commands phpunit, phpcs, phpcbf, Docker config to support PHPUnit testing. - <https://github.com/benjifisher/drupal-testing>
- The abovementioned Matt Glaman's repo from March 2021 on Drupal & Nightwatch.js training. This repository is based on a Composer build and not meant for core contributions, it is fine for contrib. - <https://github.com/bluehorndigital/drupal-testing-workshop>
- And its companion website with some details on how to get tests running: <https://bluehorndigital.github.io/drupal-testing-workshop/getting-tests-running/ddev.html>
- Generating tests with drush: <https://www.drush.org/latest/generators/>

Documentation

- Mink documentation: <https://mink.behat.org/en/latest/>

Testing setup

- Debug any of Drupal's PHPUnit tests in PhpStorm with a DDEV-Local Environment from August 3, 2021 by Joe Shindelar. Includes how to set up Chromedriver in DDEV: <https://drupalize.me/blog/debug-any-drupals-phpunit-tests-phpstorm-ddev-local-environment>

- Joe references this article from 5-13-2021 called Update 2021 - Fully integrate DDEV and PHPStorm - including Unit Tests with Coverage by \@sasunegomo <https://susi.dev/fully-integrate-ddev-and-phpstorm-including-unit-tests-with-coverage-update-2021/>
- Setup Behat for Drupal 8/9 with DDEV-Local and Selenium Recipe on Github as part of the ddev-contrib repo from July 2021 by Mike Miles: <https://github.com/drud/ddev-contrib/tree/master/docker-compose-services/drupal8-behat-selenium>
- Randy Fay lays out some details about running Selenium/Behat inside DDEV containers in August 2020 <https://stackoverflow.com/questions/51527663/running-selenium-tests-using-behat-drupal-extension-inside-ddev-containers>
- Running and debugging PHPUnit tests in PHPStorm with DDev and xdebug from 8-19-21 by Michael Strelan of Australia which includes 15 minute video showing all this. You can also checkout the [ddev-phpunit-demo](#) repo if you want to try it out yourself with DDev and PHPUnit pre-configured. <https://www.previousnext.com.au/blog/running-and-debugging-phpunit-tests-phpstorm-ddev-and-xdebug>
- Matt Glaman's Guide to Test-Driven Development with DDEV and Drupal by Heather McNamee 1-30-2019. She runs through the series of 2018 articles that Matt published
 - **Part 1.** [Running Drupal's PHPUnit test suites on DDEV](#). How to execute PHPUnit from within the web container in DDEV for Unit, Kernel, and Functional tests.
 - **Part 2.** [Running Drupal's FunctionalJavascript tests on DDEV](#). How to Chromedriver running to execute the FunctionalJavascript test suite.
 - **Part 3.** [Running Drupal's Nightwatch test suite on DDEV](#). How to run Drupal's newest testing framework: Nightwatch.js, for end-to-end tests in Node.js run against a Selenium/WebDriver server.
- Running Drupal's PHPUnit test suites on DDEV by Matt Glaman from October 2018. The first in his series referenced above <https://glamanate.com/blog/running-drupals-phpunit-test-suites-ddev>
- DDEV Contrib Repo <https://github.com/drud/ddev-contrib>

Mocking

- From Matthew Radcliffe mradcliffe \@mattkineme – Mocking Drupal: Unit Testing in Drupal 8 slides from a 2015 presentation at <http://drupalcampohio.org/sites/default/files/slides/dco2015-mocking-drupal.pdf>
 - Drupal 8/9: Unit Test cases mocking the global Drupal object and Services by Vishwa Chikate - Talks about prophecy objecting mocking which is better than PHPUnit's built in mocking. <https://medium.com/@vishwa.chikate/drupal-8-9-unit-test-cases-mocking-the-global-drupal-object-and-services-bc536477edff>
-
-

[Back to top](#)

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

Page last modified: Jun 27 2023.

[Edit this page on GitHub](#)

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

TWIG

TABLE OF CONTENTS

- [Overview](#)
 - [Theme System Overview](#)
 - [Twig Templating Engine](#)
- [Displaying Data](#)
 - [Fields or Logic](#)
 - [Which template, which variables?](#)
 - [Display fields or variables](#)
 - [Node Title with and without a link](#)
 - [Fields](#)
 - [Paragraph field](#)
 - [Loop thru paragraph reference fields](#)
 - [Body](#)
 - [Multi-value fields](#)
 - [Fields with HTML](#)
 - [The date/time a node is published, updated or created](#)
 - [Format a date field](#)
 - [Smart date field formatting](#)
 - [Entity Reference field](#)
 - [Entity reference destination content](#)
 - [Taxonomy term](#)
 - [Render a block](#)
 - [Render a list created in the template_preprocess_node\(\)](#)
 - [Links](#)
 - [Links to other pages on site](#)
 - [Link to a user using user id](#)
 - [External link in a field via an entity reference](#)
 - [Render an internal link programatically](#)
 - [Render an image with an image style](#)
 - [Hide if there is no content in a field or image](#)
 - [Hide if there is no image present](#)
 - [Attributes](#)
 - [Output the content but leave off the field_image](#)
 - [Add a class](#)
 - [Add a class conditionally](#)
 - [Links to other pages on site](#)
 - [Loop.index in a paragraph twig template](#)
 - [Loop thru an array of items with a separator](#)
- [Add Javascript into a twig template](#)
- [Control/Logic](#)
 - [Concatenate values into a string with join](#)
 - [Include partial templates](#)
 - [Loop through entity reference items](#)
 - [IF OR](#)
 - [Test if a formatted text field is empty](#)
 - [Test empty variable](#)

- [Conditionals \(empty, defined, even\)](#)
- [Test if a paragraph is empty using striptags](#)
- [Comparing strings](#)
- [Include other templates as partials](#)
- [Check if an attribute has a class](#)
- [Remove an attribute](#)
- [Convert attributes to array](#)
- [Views](#)
 - [Render a view with contextual filter](#)
 - [Count how many rows returned from a view](#)
 - [If view results empty, show a different view](#)
 - [Selectively pass 1 termid or 2 to a view as the contextual filter](#)
 - [Views templates](#)
 - [Inject variables](#)
 - [Same field used twice](#)
 - [Concatenate values into a string with join](#)
 - [Loop through entity reference items](#)
- [Twig filters and functions](#)
- [Twig Tweak](#)
 - [Documentation](#)
 - [Display a block with twig_tweak](#)
 - [Display filter form block](#)
 - [Embed view in twig template](#)
 - [Some tricky quotes magic](#)
- [Troubleshooting](#)
 - [Enable Twig debugging and disable caches](#)
 - [Debugging - Dump a variable](#)
 - [Dump taxonomy reference field](#)
 - [Using kint or dump to display variable in a template](#)
 - [502 bad gateway error](#)
 - [Views error](#)
 - [Striptags \(when twig debug info causes it to fail\)](#)
- [Reference](#)

views 602

Drupal 10 uses **Twig 3**. Drupal 9 uses Twig 2. Drupal 8 used Twig 1.

Overview

Theme System Overview

Drupal's theme system allows a theme to have nearly complete control over the appearance of the site, which includes both the markup and the CSS used to style the markup. For this system to work, instead of writing HTML markup directly, modules return "render arrays", which are structured hierarchical arrays that include the data to be rendered into HTML, and options that affect the markup. Render arrays are ultimately rendered into HTML or other output formats by recursive calls to [\Drupal\Core\Render\RendererInterface::render\(\)](#), traversing the depth of the render array hierarchy. At each level, the theme system is invoked to do the actual rendering. See the documentation of [\Drupal\Core\Render\RendererInterface::render\(\)](#) and the [Theme system and Render API topic](#) for more information about render arrays and rendering.

Twig Templating Engine

Drupal uses the templating engine Twig. Twig offers developers a fast, secure, and flexible method for building templates for Drupal 8 sites. Twig does not require front-end developers to know PHP to build and manipulate Drupal themes.

For more on theming in Drupal see <https://www.drupal.org/docs/theming-drupal>.

For further Twig documentation see <https://twig.symfony.com/doc/2.x> and <https://twig.symfony.com/doc/3.x>

Note. Drupal 10 uses **Twig 3**, Drupal 9 uses Twig 2 and Drupal 8 used Twig 1.

Displaying Data

Fields or Logic

Twig can do things that PHP can't such as whitespace control, sandboxing, automatic HTML escaping, manual contextual output escaping, inclusion of custom functions and filters that only affect templates.

Double curly braces are used to output a variable. E.g.

```
{{ content.title }}
```

Brace and percent are used to put logic into Twig templates e.g. if, then, else or for loops. E.g.

```
{% if content.price is defined %}  
  <h2>Price: {{ content.price }} </h2>  
{% endif %}
```

Use brace and pound symbol (hash) for comments e.g.

```
{# this section displays the voting details #}
```

Here are some of the Twig functions that you can use in twig templates: <https://www.drupal.org/docs/8/theming/twig/functions-in-twig-templates> There are lots of them e.g.

- file_url(\$uri)
- link(\$text, \$uri, \$attributes)
- path(\$name, \$parameters, \$options)
- url(\$name, \$parameters, \$options)

And even more Twig fun at <https://twig.symfony.com/doc/3.x/functions/index.html>

Which template, which variables?

There is usually one `page.tpl.php` and *multiple* node templates. One node template per content type. Eg. `node-news-story.html.twig`, `node-event.html.twig`. There can also be field specific templates e.g. `web/themes/custom/txg/templates/field/field--field-3-column-links.html.twig`

In the `page.html.twig`, you can refer to variables as `page.content` OR `node.label`

whereas node templates expect `content.field_image` OR `node.field_myfield`

Note. If you don't see a field output for a node, try specifying the preface `node.` instead of `content..`

Field specific template are usually very simple and refer to

```
{{ items }}
```

and

```
{{ item.content }}
```

e.g. from `txg/web/themes/contrib/zurb_foundation/templates/page.html.twig`

```
<section>  
  {{ page.content }}  
</section>
```

And from txg/web/themes/custom/tgx/templates/content/page--node--event.html.twig I accidentally started implementing this in the page template. See below for the node template.

```
{ drupal_field('field_image', 'node') }

<h1>{{ node.label }}</h1>
<div>For: {{ node.field_for.0.value }}</div>
<div>DATE: {{ node.field_event_date.0.value|date('n/j/Y') }}</div>
<div>Time: {{ node.field_event_date.0.value|date('h:i') }} - {{ node.field_event_date.0.end_value|date('h:i') }}</div>

<div>
Location:
{%
  if node.field_event_location_link.0.url %}
  <a href="{{ node.field_event_location_link.0.url }}">{{ node.field_event_location.0.value }}</a>
{%
  else %}
  {{ node.field_event_location.0.value }}
{%
  endif %}
</div>

{%
  if node.field_event_cta_link.0.url %}
  CTA:<div class="button"><a href="{{ node.field_event_cta_link.0.url }}">{{ node.field_event_cta_link.0.title }}</a></div>
{%
  endif %}
```

Here is the same basic stuff (as above) but implemented in the node template at txg/web/themes/custom/tgx/templates/content/node--event.html.twig:

Note. That `node.label` becomes `label` and `node.field_for` becomes `content.field_for`.

```
<h1>{{ label }}</h1>
{{ content.field_image }}

<div>Node: {{ node.id }}</div>
<div>For: {{ content.field_for }}</div>
<div>DATE: {{ node.field_event_date.0.value|date('n/j/Y') }}</div>
<div>Time: {{ node.field_event_date.0.value|date('h:i') }} - {{ node.field_event_date.0.end_value|date('h:i') }}</div>

<div>
Location:
{%
  if node.field_event_location_link.0.url %}
  <a href="{{ node.field_event_location_link.0.url }}">{{ node.field_event_location.0.value }}</a>
{%
  else %}
  {{ node.field_event_location.0.value }}
{%
  endif %}
</div>

{%
  if node.field_event_cta_link.0.url %}
  CTA:<div class="button"><a href="{{ node.field_event_cta_link.0.url }}">{{ node.field_event_cta_link.0.title }}</a></div>
{%
  endif %}
```

Display fields or variables

Using `node.field_myfield` will bypass the rendering and display any markup in the field. Using `content.field_myfield` uses the rendering system and is the preferred way to display your content.

This will display all the content rendered

```
{ content }
```

Node Title with and without a link

Render node title (or label) (with markup – so it may include `` tags)

```
{ label }
```

Render node label (without markup – no html in this version)

```
{node.label}
```

Render link to node

```
<a href="url">label</a>
```

// Or a little more complex..

```
<div class="title"><a href="url">label</a> | <span>content.field_vendor_ref</span></div>
```

Fields

There are many ways to limit things and only show some of the content. Mostly often you will need to show specific fields. Note. This will include rendered info such as labels etc.

```
{content.field_yomama}
```

or

```
{content.field_ref_topic}
```

Any field – just jam `content.` in front of it

```
{content.field_intl_students_and_scholars}
```

You can also grab node specific fields if `content.` type fields don't do the trick.

In a node template, you can display specific node fields by prefacing them with `node` e.g.:

```
{node.id}  
{node.label}  
{node.field_date.value}  
{node.field_date.end_value}
```

Paragraph field

These still work fine: `content.field_abc` or `node.field_ref_topic` but instead of `node`, you preface fields with `paragraph` like this:

```
termid0: {paragraph.field_ref_tax.0.target_id}  
termid1: {paragraph.field_ref_tax.1.target_id}
```

and we get this result if we have selected two terms 13 and 16.

```
termid0: 13  
termid1: 16
```

To dump a taxonomy reference field for debugging purposes use the code below. The `pre` tags format it a little nicer than if we don't have them.

```
<pre>  
{dump(paragraph.field_ref_tax.value)}  
</pre>
```

Loop thru paragraph reference fields

Here we go looping thru all the values in a multi-value reference field.

```
{% for tax in paragraph.field_ref_tax %}  
<div>target_id: {{ tax.target_id }}</div>  
{% endfor %}
```

It's the same as outputting these guys:

```
termid0: {{ paragraph.field_ref_tax.0.target_id }}  
termid1: {{ paragraph.field_ref_tax.1.target_id }}
```

and to make this more useful, here we build a string of them to pass to a view.

From: dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--news-preview.html.twig

```
{# Figure out parameters to pass to view for news items #}  
{% set params = " "%}  
{% for item in paragraph.field_ref_tax_two %}  
  {% set params = params ~ item.target_id %}  
  {% if not loop.last %}  
    {% set params = params ~ '+' %}  
  {% endif %}  
{% endfor %}  
params: {{ params }}
```

This will output something like: 5+6+19

Body

```
{{ content.body }}
```

Or

```
{{ node.body.value }}
```

And for summary

```
{{ node.body.summary | raw }}
```

Multi-value fields

Fields that you preface with `node.` can also handle an index (the `0` below) i.e. to indicate the first value in a multi-value field, `1` to indicate the second etc.

```
{{ node.field_iso_n3_country_code.0.value }}
```

Fields with HTML

If a field has html that you want rendered, use the keyword `raw`. Be aware this has security considerations which you can mitigate using [striptags](#) filters:

```
<div>How to order: {{ how_to_order|raw }}</div>
```

And maybe you want to only allow `` tags

```
{{ word|striptags('<b>')|raw }}
```

Or several tags. In this case `<a><pre>`

```
{{ word|striptags('<b>,<a>,<pre>')|raw }}
```

The date/time a node is published, updated or created

Each of these calls return a datetime value in string form which can be massaged by the twig date() function for formatting.

```
<pre>
Created: {{ node.created.value }}
Created: {{ node.createdtime }}
Created: {{ node.created.value|date('Y-m-d') }}

Modified: {{ node.changed.value }}
Modified: {{ node.changedtime }}
Modified: {{ node.changed.value|date('Y-m-d') }}

Published: {{ node.published_at.value }}
Published: {{ node.published_at.value|date('Y-m-d') }}
</pre>
```

Here is the output you might see. Note. The first published is apparently blank because I didn't use the drupal scheduling to publish the node (maybe?) and the second one seems to have defaulted to today's date.

```
Created: 1604034000
Created: 1604034000
Created: 2020-10-30
Modified: 1604528207
Modified: 1604528207
Modified: 2020-11-04
Published:
Published: 2020-11-20
```

Updated/changed

```
{% set post_date = node.changedtime %}
```

Created (same as authored on date on node edit form):

```
 {{ node.createdtime }}
```

And pretty formatted like Sep 2, 2023

```
 {{ node.createdtime|date('M d, Y') }}
```

Also

```
<div class="date">Date posted: {{ node.getCreatedTime|date('m/d/Y') }}</div>
<div class="date">Date posted: {{ node.getChangedTime|date('m/d/Y') }}</div>
```

Node published date:

```
Date published: {{ _context.node.published_at.value }}
Date published: {{ node.published_at.value }}
```

Format a date field

Use the field's format settings; include wrappers. This example includes wrappers.

```
 {{ content.field_blog_date }}
```

The examples below do not include wrappers.

Use the field's format settings. This will use the format defined in Content type » Manage Displays »Your View Mode.

```
{% set start = node.field_blog_date.0 %}
```

Using Twig date filter and a defined Drupal date format

```
{% set start = node.field_blog_date.value|date('U')|format_date('short_mdyyyy') %}
```

Use Twig date filter

```
{% set start = node.field_blog_date.value|date('n/j/Y') %}
```

Smart date field formatting

When using the [smart_date](#) module, dates are stored as timestamps so you have to use the twig date function to format them. If you just put this in your template:

```
{% set start = node.field_when.0 %}
```

The output will include whichever formatting you specify in Drupal. While I assume there is a way to pass a [smart_date](#) formatting string to twig, I haven't discovered it yet. Here are ways to format a [smart_date](#).

Specify the index (the 0 indicating the first value, or 1 for the second) e.g. node.field.0.value and pipe the twig [date](#) function for formatting:

Date as in July 18, 2023

```
{% set start = node.field_when.0.value|date('F j, Y') %}
```

End date

```
{% set end = node.field_when.0.end_value|date('F j, Y') %}
```

Timezone as in America/Chicago

```
{% set start = node.field_when.0.value|date('e') %}
```

Timezone as in CDT

```
{% set start = node.field_when.0.value|date('T') %}
```

Day of the week

```
{% set start = node.field_when.0.value|date('l') %} {# day of week #}
```

Hide the end date if it is the same as the start date

```
{% set start = node.field_when.0.value|date('l F j, Y') %}
{% set end = node.field_when.0.end_value|date('l F j, Y') %}
<p class="date">{{ start }}</p>
{% if not start is same as(end) %}
<p class="date">{{ end }}</p>
{% endif %}
```

Entity Reference field

If you have an entity reference field such as field_ref_topic (entity reference to topic content) you have to specify the target_id like this. If you have only 1 reference, use the .0, for the second one use .1 and so on.

```
 {{ node.field_ref_topic.0.target_id }}
```

Note. This will show the node id of the entity reference field. See below to see the content that the entity reference field points to.

Entity reference destination content

If you have an entity reference and you want to display the content from the node that is referenced i.e. if you have a contract with a reference to the vendor node and you want to display information from the vendor node on the contract you can dereference fields in the entity destination:

From `dirt/web/themes/custom/dirt_bootstrap/templates/content/node--contract--vendor-list.html.twig`:

```
 {{ node.field_sf_contract_ref.entity.field_contract_overview.value }}
```

Or

```
 {{ content.field_sf_contract_ref.entity.field_contract_overview }}
```

The field in the contract node is called `field_sf_contract_ref`. The field in the referenced entity is called `field_contract_overview`. Notice how with the `node` style, you must specify `.value` at the end.

Here is an example of a taxonomy term where the title of the term will be displayed.

```
<pre>
Dump category:
{{ dump(node.field_ref_tax.entity.label) }}
</pre>
```

Taxonomy term

Here is an example of displaying a taxonomy term.

```
<pre>
Dump category: {{ dump(node.field_ref_tax.entity.label) }}
</pre>
```

Render a block

Example block with a machine name of `block--system-powered-by-block.html.twig` from a custom theme

```
{%
  set classes = [
    'block',
    'block-' ~ configuration.provider|clean_class,
    'block-' ~ plugin_id|clean_class,
  ]
%}
<div{{ attributes.addClass(classes) }}>
  {{ title_prefix }}
  {% if label %}
    <h2{{ title_attributes }}>{{ label }}</h2>
  {% endif %}
  {{ title_suffix }}
  {% block content %}
    {{ content }}
  {% endblock %}
  also powered by <a href="http://austinprogressivecalendar.com">Austin Progressive Calendar</a>
</div>
```

Render a list created in the `template_preprocess_node()`

Here we create a list in the function:

```
function burger_theme_preprocess_node(&$variables) {  
  
  $burger_list = [  
    ['name' => 'Cheesburger'],  
    ['name' => 'Mushroomburger'],  
    ['name' => 'Chickenburger'],  
  ];  
  $variables['burgers'] = $burger_list;  
}
```

and render it in the node--article--full.html.twig

```
<ol>  
  {% for burger in burgers %}  
    <li>{{ burger['name'] }}</li>  
  {% endfor %}  
</ol>
```

Links

There are a bajillion kertrillion or more ways to render a link

Link field (URL)

This is the simplest way. Just set the display mode to link



And output the link without a label.

```
{{ content.field_suggest_button }}
```

If you need a little more control you might use this version which allows classes etc. We are adding several classes onto the anchor to make it look like a button. In this case with an internal link, it shows up using the alias of the link i.e. it shows /contracts instead of node/7 when you hover over the link.

```
<p><a class="btn secondary navy centered" href="{{ node.field_suggest_button.0.url }}">{{ node.field_suggest_button.0.title }}</a></p>
```

Using .uri causes the link (internal only. External links are fine) to show up as node/7 when you hover over the link.

```
<p><a class="btn secondary navy centered" href="{{ node.field_suggest_button.uri }}">{{ node.field_suggest_button.0.title }}</a></p>
```

Don't try this as it won't work:

```
//bad  
{{ node.field_suggest_button.url }}.  
//bad
```

Want to use the text from a different field? No problem.

```
<div class="title"><a href="{{ node.field_link.uri }}">{{ node.field_contract_number.value }}</a></div>
```

Links to other pages on site

Absolute link:

```
<a href="{{ url('entity.node.canonical', {node: 3223}) }}>Link to Weather Balloon node 3223 </a>
```

Relative link

See path vs url:

```
<a href="{{ path('entity.node.canonical', {node: 3223}) }}>Link to Weather Balloon node 3223 </a>
```

Link to a user using user id

You can link to users using the following:

```
<a href="{{ url('entity.user.canonical', {user: 1}) }}>Link to user 1 </a>
```

External link in a field via an entity reference

Here we have a node with an entity reference field (`field_sf_contract_ref`) to another entity.

In a preprocess function, you can grab the link. Note, you can just grab the `first()` one. Later on you can see that in the twig template, you can specify the first one with `.0`

From `dirt/web/themes/custom/dirt_bootstrap/dirt_bootstrap.theme`

```
$vendor_url = $node->field_sf_contract_ref->entity->field_vendor_url->first();
if ($vendor_url) {
  $vendor_url = $vendor_url->getUrl();
  if ($vendor_url) {
    $variables['vendor_url'] = $vendor_url->getUri();
  }
}
```

And in the template we retrieve the URI with `.uri`:

```
<p><a class="styled-link ext" href="{{ node.field_sf_contract_ref.entity.field_vendor_url.uri }}>Vendor Website</a></p>
```

Here we check if there is a target value and output that also. E.g. `target="_blank"` and also display the title – this is the anchor title as in the words "Vendor Website" below

```
<a href="https://www.duckduckgo.com">Vendor Website</a></p>
```

From `inside-marthe/themes/custom/dp/templates/paragraph/paragraph--sidebar-product-card.html.twig` we wrap some stuff in a link:

```
<a href="{{ content.field_link[0]['url'] }}" {%
  if content.field_link[0]['options']['attributes']['target'] %} target="{{ content.field_link[0]['options']['attributes']['target'] }}"
  class="module-link"
  {{ content.field_image }}
  <h2 class="module-header">{{ content.field_text }}</h2>
  {{ content.field_text2 }}
}>
```

And from `txg/web/themes/custom/tgx/templates/content/node--event--card.html.twig` if there is a url, display the link with the url, otherwise just display the title for the link. I'm not 100% sure this is really valid. Can you put in a title and no link?

```

{% if node.field_event_location_link[0].url %}
<a href="{{ node.field_event_location_link[0].url }}>{{ node.field_event_location[0].value }}</a>
{% else %}
{{ node.field_event_location[0].value }}
{% endif %}

```

Render an internal link programatically

Here we want to render an internal link to a page on our Drupal site (as opposed to a link to another site.) We grab the link in a preprocess function. Extract out the title and the URI.

```

$instructions_node = Node::load($order_type_instructions_nid);
if ($instructions_node) {
  $order_link = $instructions_node->field_link->first();
  if ($order_link) {
    $uri = $order_link->uri;
    $variables['order_link_title'] = $order_link->title;
    $order_url = $order_link->getUrl();
    if ($order_url) {
      $variables['order_type_link'] = $order_url;
    }
  }
}

```

We can put the pieces in the twig template like this

```
<a href="{{ order_type_link }}>{{ order_link_title }}</a>
```

Render an image with an image style

From inside-marthe/themes/custom/dp/templates/paragraph/paragraph--sidebar-resource.html.twig

Here we use sidebar_standard image style

```
<aside class="module module--featured" data-interchange="[[ content.field_image[0]['#item'].entity.uri.value | image_style('sidebar_standard') ]], small">
```

Or for a media field, set the image style on the display mode and use this:

```
 {{ content.field_banner_image[0] }}
```

Hide if there is no content in a field or image

From inside-marthe/themes/custom/dp/templates/content/node--video-detail.html.twig I check to see if there are any values in this array related_lessons_nids and display the view.

```
{% if related_lessons_nids|length %}

<div class="section section--featured">
  <div class="grid-container">
    <h2 class="section-header text-center large-text-left">Related Lessons</h2>
    <div class="grid-x grid-margin-x" data-equalizer data-equalize-on="large">
      {{ drupal_view('video', 'embed_collection_related_lessons', related_lessons_nids|join(', ')) }}
    </div>
  </div>
</div>

{% endif %}
```

Not empty:

```
{% if content.field_myfield is not empty %}
  {# Do something here #}
{% endif %}
```

Hide if there is no image present

If there is an image (and it is renderable) display the image

```
{% if content.field_teacher_commentary_image|render %}
  
{% endif %}
```

Attributes

From <https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes>:

Elements in HTML have **attributes**; these are additional values that configure the elements or adjust their behavior in various ways to meet the criteria the users want.

Read more about using attributes in templates <https://www.drupal.org/docs/8/theming-drupal-8/using-attributes-in-templates>

To add a data attribute use:

```
  {{ attributes.setAttribute('data-myname','tommy') }}
```

e.g.

```
<article{{ attributes.addClass(classes)|setAttribute('my-name', 'Selwyn') }}>
```

Produces:

```
<article
  data-history-node-id="3224"
  data-quicredit-entity-id="node/3224"
  role="article"
  class="contextual-region node node--type-article node--promoted node--view-mode-full"
  about="/burger1"
  typeof="Schema:Article"
  my-name="Selwyn"
  data-quicredit-entity-instance-id="0"
></article>
```

More useful examples at <https://www.drupal.org/docs/8/theming-drupal-8/using-attributes-in-templates> such as:

```

{%
set classes =['red', 'green', 'blue'] %}
{%
set my_id = 'specific-id' %}
{%
set image_src = 'https://www.drupal.org/files/powerd-blue-135x42.png' %}

<img{{ attributes.addClass(classes)|removeClass('green')|setAttribute('id', my_id)|setAttribute('src', image_src) }}>

```

Which outputs the following:

```



```

Check if an attribute has a class

```

{{ attributes.hasClass($class) }}

```

Remove an attribute

```

{{ attributes.removeAttribute() }}

```

Convert attributes to array

```

{{ attributes.toArray() }}

```

Output the content but leave off the field_image

From very/web/themes/very/templates/node--teaser.html.twig:

```

<div{{ content_attributes.addClass('content') }}>
  {{ content|without('field_image')|render|striptags }}
</div>

```

Add a class

```

<div{{ content_attributes.addClass('node__content') }}>

```

Add a class conditionally

From very/web/themes/very/templates/node--teaser.html.twig

For an unpublished node, wrap this class around the word unpublished

```

{%
if not node.published %}
  <p class="node--unpublished">{{ 'Unpublished'|t }}</p>
{%
endif %}

```

Links to other pages on site

Absolute:

```

<a href="{{ url('entity.node.canonical', {node: 3223}) }}">Link to WEA node 3223 </a>

```

Relative (see path vs url):

```

<a href="{{ path('entity.node.canonical', {node: 3223}) }}">Link to WEA node 3223 </a>

```

Could also link to users using

```
<a href="{{ url('entity.user.canonical', {user: 1}) }}>Link to user 1 </a>
```

Loop.index in a paragraph twig template

From: web/themes/custom/dprime/templates/field/field--paragraph--field-links--sidebar-cta.html.twig

Notice the use of `loop.index` to only output this for the first item

```
{% for item in items %}  
  {% if loop.index == 1 %}  
    <div class="cell medium-6">  
      <a href="{{ item.content['#url'] }}" class="button {% if loop.index == 2 %}hollow {% endif %}button--light m-b-0" {% if item.content['#options']['target'] %} target="{{ item.content['#options']['target'] }}"{% endif %}>{{ item.content['label'] }} </a>  
    </div>  
  {% endif %}  
{% endfor %}
```

Loop thru an array of items with a separator

This loads all the authors and adds and between them except for the last one:

```
<div>  
  {% if content.author -%}  
    by  
    {% for author in content.author -%}  
      {% if loop.last %}  
        {% set separator = " " %}  
      {% else %}  
        {% set separator = ' and ' %}  
      {% endif %}  
      {{ author }} {{ separator }}  
    {% endfor -%}  
  {% endif -%}  
</div>
```

This version inserts commas:

From: org/docroot/themes/custom/org/templates/field/field--node--field-categories--opinion.html.twig

```
{% if label_hidden %}  
  {% if multiple %}  
    {% for item in items %}  
      {% if loop.index > 1 %}, {% endif %}{{ item.content }}  
    {% endfor %}  
  {% else %}  
    {% for item in items %}  
      {% if loop.index > 1 %}, {% endif %}{{ item.content }}  
    {% endfor %}  
  {% endif %}  
  {% else %}  
    <div{{ title_attributes }}>{{ label }}</div>  
    {% for item in items %}  
      {% if loop.index > 1 %}, {% endif %}{{ item.content }}  
    {% endfor %}  
  {% endif %}
```

Add Javascript into a twig template

```

<script>
function hidePRSnippet(config, data) {
  if (data.average_rating < 3.5 && data.average_rating !== 0) {
    document.getElementById('pr-reviewsnippet').remove();
  }
}

var powerReviewsConfig = {{ accPowerreviews | raw }};
for (var i = 0; i < powerReviewsConfig.length; i++) {
  if (powerReviewsConfig[i].hasOwnProperty('components') && powerReviewsConfig[i].components.hasOwnProperty('ReviewSnippet')) {
    powerReviewsConfig[i]['on_render'] = hidePRSnippet;
  }
}
POWERREVIEWS.display.render(powerReviewsConfig);
</script>

```

Control/Logic

Concatenate values into a string with join

This would typically be used when passing a series of node id's to a view to filter its output.

```
{% set blah = [node.field_ref_unit.0.target_id,node.field_ref_unit.1.target_id,node.field_ref_unit.2.target_id,node.field_ref_unit.3.target_id]join('+') %}
```

This produces 1+2+3+4

Include partial templates

```
{% include '@txg/partials/searchfilterform.html.twig' %}
```

Loop through entity reference items

In txg/web/themes/custom/txg/templates/content/node--news-story.html.twig I need to loop through a bunch of entity reference values and build a string of id+id+id... (with an undefined number) so

```

{% set blah = " " %}
{% for item in node.field_ref_unit %}
  {% set blah = blah ~ item.target_id %}
  {% if not loop.last %}
    {% set blah = blah ~ '+' %}
  {% endif %}
{% endfor %}

<div>blah:{{ blah }}</div>
<div>node id: {{ node.id }}</div>
{{ drupal_view('related_news_for_news_story', 'block_unit', node.id, blah)}}

```

IF OR

If there is a value in field_event_date or field_display_date, then display it/them.

```

<div{{ content_attributes.addClass('teaser__content') }}>
  {% if content.field_event_date or content.field_display_date %}
    <div class="teaser__date">
      {{ content.field_event_date|render|striptags }}
      {{ content.field_display_date|render|striptags }}
    </div>
  {% endif %}

```

Test if a formatted text field is empty

To check a body field or other formatted text field, use |render to render it first.

```
{% if content.body|render %}  
<li><a class="scroll" href="#section-overview">Overview</a></li>  
{% endif %}
```

Test empty variable

This code checks if a variable is empty using `empty` test if the attributes variable is not set. From

<https://www.drupal.org/project/drupal/issues/2558079>:

```
{% if attributes is empty %}  
{{ link(item.title, item.url) }}  
{% else %}  
{{ link(item.title, item.url, attributes) }}  
{% endif %}
```

You can also use:

```
{% if blah is not empty %}  
{{content.name}}  
{% endif %}
```

Conditionals (empty, defined, even)

```
{% if rows %}  
{{rows}}  
{% elseif empty %}  
{{ empty }}  
{% endif %}  
  
{% if var is defined %}  
{{ content.name }}  
{% endif %}  
  
{% if var is even %}  
{{ content.name }}  
{% endif %}
```

e.g. from inside-marthe/themes/custom/dp/templates/paragraph/paragraph--highlight-card.html.twig

```

{% set showCat = TRUE %}
{% if view_mode == 'overview' or view_mode == 'home' %}
  {% set showCat = FALSE %}
{% endif %}

<div class="cell medium-4 home-highlight-card m-b-3" data-equalizer-watch>
  <a href="{% spaceless %}{{ content.field_link.0 }}{% endspaceless %}" class="card h-100" data-interchange="[{ content.field_image.0['#item']entity.uri.value | image_style('high') }]">
    <div class="card-content">
      {% if showCat %}
        <span class="card-label">{{content.field_text2}}</span>
      {% endif %}
      <h3 class="card-header">{{content.field_text}}</h3>
      {{content.field_description}}
      {% if content.field_button_text|render %}
        <span class="button full-width show-on-hover">{{ content.field_button_text }}</span>
      {% else %}
        <span class="button full-width show-on-hover">{{ content.field_text|render }}</span>
      {% endif %}
    </div>
  </a>
</div>

```

Test if a paragraph is empty using striptags

From /inside-marthe/themes/custom/dp/templates/content/node--video-collection.html.twig:

Normally you wouldn't need the striptags, but when twig debugging is enabled, the render information includes debug tags. See <https://www.drupal.org/project/drupal/issues/2547559#comment-12103048>

```

{% if content.field_related_lessons|render|striptags|trim is not empty %}
  {{ content.field_related_lessons}}
{% endif %}

```

Or this much simpler version which also comes from the same issue page above (it doesn't seem to work as well as the version above):

```

{% if content.field_related_lessons.value %}
  {{ content.field_related_lessons}}
{% endif %}

```

Comparing strings

For complicated strings, you have to use the Twig [same as](#) function because using `if x == y` doesn't work. See the commented out part where I tried `==`:

```

{% set start = node.field_when.0.value|date('l F j, Y') %}
{% set end = node.field_when.0.end_value|date('l F j, Y') %}
<p class="date"> {{ start }}</p>
{#{% if not start == end %}#
{% if not start is same as(end) %}
  <p class="date"> {{ end }}</p>
{% endif %}

```

Include other templates as partials

In very/web/themes/very/templates/node--featured.html.twig

You can re-use templates. Just put them in the partials directory (you don't have to but it is a good convention) and include them.

```

{{ include('node--teaser.html.twig') }}

```

Check if an attribute has a class

```
{% if attributes.hasClass($class) %}
```

Remove an attribute

```
{% attributes.removeAttribute0 %}
```

Convert attributes to array

```
{% attributes.toArray 0 %}
```

Views

Render a view with contextual filter

Pro tip: Create embed displays (rather than blocks or pages) so users don't see these blocks appearing in the block management page. see <https://drupal.stackexchange.com/questions/287209/what-does-the-embed-display-type-do>

To use a field value in a view as an argument, using [twig_tweak](#), you can render the view and its arguments/parameters. In the example below, these are the contextual filters defined in the view.

```
{% drupal_view('map_data_for_a_country', 'block_stats', node.field_iso_n3_country_code.0.value) %}
```

Note. Using content.field as a parameter doesn't work because content.fields get rendered so they are usually filled with HTML or labels or both. Parameters need to simply be numbers or strings.

Other examples. Here an entity reference field is passed as a parameter. This works for taxonomy terms like this also.

```
{% drupal_view('news_stories_for_a_topic','block_1', node.field_ref_topic.0.target_id) %}
```

Or

```
{% drupal_view('resellers_for_this_vendor', 'embed_1', node.field_vendor_id.value) %}
```

Note. If you ever see a 502 bad gateway error when embedding a drupal_view, delete the display and create a new one and it may just work fine.

Count how many rows returned from a view

<https://www.drupal.org/docs/8/modules/twig-tweak/twig-tweak-and-views>

Check if View has Results

```
{% set view = drupal_view_result('related', 'block_1')|length %}
{% if view > 0 %}
  {{ drupal_view('related', 'block_1') }}
{% endif %}
```

If view results empty, show a different view

In txg/web/themes/custom/txg/templates/content/node--news-story.html.twig we show units (the first view) but if there aren't any, show aofs (the second view.)

```
{% if drupal_view_result('related_news_for_news_story', 'block_unit', node.id, unit_ids) %}
  {{ drupal_view('related_news_for_news_story', 'block_unit', node.id, unit_ids) }}
{% elseif drupal_view_result('related_news_aof', 'block_aof', node.id, aof_ids) %}
  {{ drupal_view('related_news_aof', 'block_aof', node.id, aof_ids) }}
{% endif %}
```

Selectively pass 1 termid or 2 to a view as the contextual filter

In the view, you can allow multiple terms for a contextual filter

From: <https://drupal.stackexchange.com/questions/78701/views-multiple-contextual-filters-taxonomy>:

Instead of Content: The name of Taxonomy (taxonomy_vocabulary_#) you need to select Content: Has taxonomy term ID *Contextual filter* and enable *Allow multiple values* to able to use multiple values in the form of 1+2+3 (for OR) or 1,2,3 (for AND).

Then in the template, check if there is a second value and build the arguments in the form id+id (e.g. "13+16") In this example, I have to assume the setup allows only 2 taxonomy terms to be entered. See below for an unlimited amount of terms.

From /Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--upcoming-events.html.twig:

```
{# if there is a second category, pass it separated by + #}
{% if paragraph.field_ref_tax.1.target_id %}
  {% set args = paragraph.field_ref_tax.1.target_id ~ '+' ~ paragraph.field_ref_tax.1.target_id %}
  args: {{ args }}
  {{ drupal_view('events', 'embed_2', paragraph.field_ref_tax.0.target_id ~ '+' ~ paragraph.field_ref_tax.1.target_id) }}
{% else %}
  {{ drupal_view('events', 'embed_2', paragraph.field_ref_tax.0.target_id) }}
{% endif %}
```

Or even nicer, we could loop thru an unlimited number of terms, build a string of them to pass to a view.

From: /Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--news-preview.html.twig

```
{# Figure out parameters to pass to view for news items #}
{% set params = "" %}
{% for item in paragraph.field_ref_tax_two %}
  {% set params = params ~ item.target_id %}
  {% if not loop.last %}
    {% set params = params ~ '+' ~ %}
  {% endif %}
{% endfor %}
params: {{ params }}
```

This will output something like: 5+6+19

And pass the output to a view like this:

```
{{ drupal_view('news', 'embed_2', params) }}
```

Views templates

Using machine names for the view, you can copy the base view templates (just like in Drupal 7) to make specific templates.

From <https://www.drupal.org/docs/theming-drupal/twig-in-drupal/twig-template-naming-conventions>:

Using a View named foobar with its style: unformatted and row style: Fields, and using display:Page.

```

views-view--foobar--page.html.twig
views-view--page.html.twig
views-view--foobar.html.twig
views-view.html.twig

views-view-unformatted--foobar--page.html.twig
views-view-unformatted--page.html.twig
views-view-unformatted--foobar.html.twig
views-view-unformatted.html.twig

views-view-fields--foobar--page.html.twig
views-view-fields--page.html.twig
views-view-fields--foobar.html.twig
views-view-fields.html.twig

```

or for views-view-list.html.twig, you could use views-view-list--foobar.html.twig

e.g. ./Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/views/views-view-list--resource-library.html.twig

Inject variables

You can inject variables into a view using `hook_preprocess_views_view()` eg. from `txg/web/themes/custom/txg/txg.theme`. The code below used to was load up various items to populate the select dropdown controls in the view:

```

function txg_preprocess_views_view(&$variables) {
  $view = $variables['view'];
  $id = $view->storage->id();
  $display_id = $view->current_display;

  // Build /newsroom/search
  if ($id == 'news_events_search' && $display_id == 'page_news') {
    $variables['filter_data'] = generate_search_filter_data();
  }
}

```

Here is the code that builds the data for the select controls:

```

/**
 * Populate dropdowns.
 *
 * Builds the search filter data to populate select dropdowns on
 * /newsroom, /newsroom-search etc. pages.
 *
 * @return array
 *   Array of values for dropdown.
 *
 * @throws \Drupal\Component\Plugin\Exception\InvalidPluginDefinitionException
 * @throws \Drupal\Component\Plugin\Exception\PluginNotFoundException
 */
function generate_search_filter_data() {

  // Grab the Get parameters.
  $country_nid = \Drupal::request()->query->get('country');
  $aof_nid = \Drupal::request()->query->get('aof');
  $unit_nid = \Drupal::request()->query->get('unit');
  $continent_arg = \Drupal::request()->query->get('continent');
  $topic_tid = \Drupal::request()->query->get('topic');

  // Office: AOF and child units from main menu.
  $office_nid = 0;
}

```

```

if (!empty($aof_nid)) {
  if (is_numeric($aof_nid)) {
    $office_nid = $aof_nid;
  }
}
if (!empty($unit_nid)) {
  if (is_numeric($unit_nid)) {
    $office_nid = $unit_nid;
  }
}

$storage = get_offices($office_nid);
$data[] = [
  'type' => 'office',
  'info' => $storage,
];

```

// Topic taxonomy terms.

```

$vid = 'topic';
$terms = \Drupal::entityTypeManager()->getStorage('taxonomy_term')->loadTree($vid);
$storage = [];
foreach ($terms as $term) {
  $storage[] = [
    "value" => $term->tid,
    "title" => $term->name,
  ];
}

```

// Update the select to show the current value from the url.

```

foreach ($storage as &$item) {
  if ($item['value'] == $topic_tid) {
    $item['selected'] = 'selected';
    break;
  }
}

```

```

$data[] = [
  'type' => 'topic',
  'info' => $storage,
];

```

// Continent.

```

$continents = [
  ['title' => 'Africa', 'value' => 'Africa'],
  ['title' => 'Antarctica', 'value' => 'Antarctica'],
  ['title' => 'Asia', 'value' => 'Asia'],
  ['title' => 'Europe', 'value' => 'Europe'],
  ['title' => 'North America', 'value' => 'north%20america'],
  ['title' => 'Oceania', 'value' => 'Oceania'],
  ['title' => 'South America', 'value' => 'south%20america'],
];

```

// Update the select to show the current value from the url.

```

foreach ($continents as &$continent) {
  if (!empty($continent_arg)) {
    if (strtolower($continent['title']) == strtotime($continent_arg)) {
      $continent['selected'] = 'selected';
    }
  }
}

```

```

$data[] = [
  'type' => 'continent',

```

```

'info' => $continents,
];

// Country.

$data_storage = Drupal::entityTypeManager()->getStorage('node');
$query = Drupal::entityQuery('node')
->condition('status', 1)
->condition('type', 'country')
->sort('title', 'ASC');
$nids = $query->execute();
$nodes = $data_storage->loadMultiple($nids);
$storage = [];
foreach ($nodes as $node) {
  $storage[] = [
    'title' => $node->getTitle(),
    'value' => $node->id(),
  ];
}

// Set the default country so it appears in the select dropdown.

foreach ($storage as &$item) {
  if ($item['value'] == $country_nid) {
    $item['selected'] = 'selected';
    break;
  }
}
$data[] = [
  'type' => 'country',
  'info' => $storage,
];
}

return $data;
}

```

Here the output for all views uses the `views-view.html.twig` template

```
<!-- BEGIN OUTPUT from 'core/themes/classy/templates/views/views-view.html.twig' -->
```

If we want to override the `frontpage` view we can copy the template from above to our theme and rename it `views-view--frontpage.html.twig`

Notice that it will override all displays (in this case the `page` and the `feed` displays – “`page_1`” and “`feed_1`” respectively) so we can be more specific

Rename it to `views-view--frontpage--page_1.html.twig`

From https://api.drupal.org/api/drupal/core%21modules%21views%21views.theme.inc/group/views_templates/8.5.x

All views templates can be overridden with a variety of names, using the view, the display ID of the view, the display type of the view, or some combination thereof.

For each view, there will be a minimum of two templates used. The first is used for all views: [views-view.html.twig](#).

The second template is determined by the style selected for the view. Note that certain aspects of the view can also change which style is used; for example, arguments which provide a summary view might change the style to one of the special summary styles.

The default style for all views is [views-view-unformatted.html.twig](#).

Many styles will then farm out the actual display of each row to a row style; the default row style is `views-view-fields.html.twig`.

Here is an example of all the templates that will be tried in the following case:

View: foobar Style: unformatted Row style: Fields. Display:Page.

```
- views-view--foobar--page.html.twig
- views-view--page.html.twig
- views-view--foobar.html.twig
- views-view.html.twig
- views-view-unformatted--foobar--page.html.twig
- views-view-unformatted--page.html.twig
- views-view-unformatted--foobar.html.twig
- views-view-unformatted.html.twig
- views-view-fields--foobar--page.html.twig
- views-view-fields--page.html.twig
- views-view-fields--foobar.html.twig
- views-view-fields.html.twig
```

IMPORTANT

When adding a new template to your theme, be sure to flush the theme registry cache!

From

https://api.drupal.org/api/drupal/core%21modules%21views%21views.theme.inc/function/template_preprocess_views_view_field/8.2.x

When changing the value of a field in a view, use `preprocess_views_view_field()`.

```
function mytheme_preprocess_views_view_field(&$variables) {
  $view = $variables['view'];
  if($view->id() == 'my_view') {
    if($variables['field']->field == 'field_my_field') {
      $my_field_value = $variables['field']->getValue($variables['row']);
      $my_altered_value = 'xx';
      $variables['output'] = $my_altered_value; // Default variable in Twig file is "output"
      // OR
      $variables['my_output'] = $my_altered_value; // New variable in Twig file (views-view-field--my-view--field-my-field.html.twig)
    }
  }
}
```

SAME FIELD USED TWICE

Note. If you use the same field twice in a view i.e. if you need to display different parts of the same field in different places, views names them something like this: `field_library_media` and `field_library_media_1`. In that circumstance, you have to refer to them in the function like this:

```
// if($variables['field']->field == 'field_library_media') { if($variables['field']->options['id'] == 'field_library_media_1') {
```

Here is a real example where a media field id is being displayed and I switch it out with the formatted size of the media file.

```

/**
 * Implements hook_preprocess_views_view_field().
 */
function dirt_bootstrap_preprocess_views_view_field(&$variables) {
  $view = $variables['view'];
  if($view->id() == 'resource_library') {
    // if($variables['field']->field == 'field_library_media') {
      if($variables['field']->options['id'] == 'field_library_media_1') {
        $target_id = $variables['field']->getValue($variables['row']);
        $file_size = 0;
        if ($target_id) {
          $media_item = Media::load($target_id);
          // Get the file.
          if ($media_item->hasField('field_media_document')) {
            $file_id = $media_item->field_media_document->getValue()[0]['target_id'];
          }
          elseif ($media_item->hasField('field_media_image')) {
            $file_id = $media_item->field_media_image->getValue()[0]['target_id'];
          }
          elseif ($media_item->hasField('field_media_audio_file')) {
            $file_id = $media_item->field_media_audio_file->getValue()[0]['target_id'];
          }
          elseif ($media_item->hasField('field_media_video_file')) {
            $file_id = $media_item->field_media_video_file->getValue()[0]['target_id'];
          }
          if (isset($file_id)) {
            $file = File::load($file_id);
            if ($file) {
              // Get file size.
              $file_size = format_size($file->getSize());
            }
          }
        }
        $variables['output'] = $file_size; // Default variable in Twig file is "output"
      }
    }
  }
}

```

Concatenate values into a string with join

This would typically be used when passing a series of node id's to a view to filter its output.

```
{% set blah = [node.field_ref_unit.0.target_id,node.field_ref_unit.1.target_id,node.field_ref_unit.2.target_id,node.field_ref_unit.3.target_id]|join('+') %}
```

This produces 1+2+3+4

Loop through entity reference items

In txg/web/themes/custom/tgx/templates/content/node--news-story.html.twig I need to loop through a bunch of entity reference values and build a string of id+id+id... (with an undefined number) so:

```

{%
set blah = ""
for item in node.field_ref_unit %}
{%
set blah = blah ~ item.target_id %}
{%
if not loop.last %}
{%
set blah = blah ~ '+' %}
{%
endif %}
{%
endfor %}

<div>blah:{{ blah }}</div>
<div>node id: {{ node.id }}</div>
{{ drupal_view('related_news_for_news_story', 'block_unit', node.id, blah) }}

```

Twig filters and functions

Use these to do almost anything with variables in your twig templates.

See cheat sheet at <https://www.drupal.org/docs/contributed-modules/twig-tweak/cheat-sheet#s-view-filter> also <https://twig.symfony.com/doc/3.x/> for filters and functions

Here are some examples. A complete list is included below:

```

{{ <span>Hello I am an html twig, but my html will be stripped</span> | striptags }}

{{'welcome' | upper }}

{{ data | json_encode() }}

{%
filter upper %}
This test becomes uppercase
{%
endfilter %}

```

Filters

- [abs](#)
- [batch](#)
- [capitalize](#)
- [column](#)
- [convert_encoding](#)
- [country_name](#)
- [currency_name](#)
- [currency_symbol](#)
- [data_uri](#)
- [date](#)
- [date_modify](#)
- [default](#)
- [escape](#)
- [filter](#)
- [first](#)
- [format](#)
- [format_currency](#)
- [format_date](#)
- [format_datetime](#)
- [format_number](#)
- [format_time](#)
- [html_to_markdown](#)
- [inky_to_html](#)
- [inline_css](#)

- [join](#)
- [json_encode](#)
- [keys](#)
- [language_name](#)
- [last](#)
- [length](#)
- [locale_name](#)
- [lower](#)
- [map](#)
- [markdown_to_html](#)
- [merge](#)
- [nl2br](#)
- [number_format](#)
- [raw](#)
- [reduce](#)
- [replace](#)
- [reverse](#)
- [round](#)
- [slice](#)
- [slug](#)
- [sort](#)
- [spaceless](#)
- [split](#)
- [striptags](#)
- [timezone_name](#)
- [title](#)
- [trim](#)
- [u](#)
- [upper](#)
- [url_encode](#)

Functions

- [attribute](#)
- [block](#)
- [constant](#)
- [country_names](#)
- [country_timezones](#)
- [currency_names](#)
- [cycle](#)
- [date](#)
- [dump](#)
- [html_classes](#)
- [include](#)
- [language_names](#)
- [locale_names](#)
- [max](#)
- [min](#)
- [parent](#)
- [random](#)
- [range](#)
- [script_names](#)
- [source](#)

- [template_from_string](#)
- [timezone_names](#)

Twig Tweak

This is an **essential** module to add to all projects.

Documentation

There are some docs at <https://www.drupal.org/docs/contributed-modules/twig-tweak/rendering-blocks-with-twig-tweak>

and a cheat sheet at https://git.drupalcode.org/project/twig_tweak/-/blob/3.x/docs/cheat-sheet.md

Display a block with twig_tweak

Here is a simple example:

```
{{ drupal_block('plugin_id') }}
```

It looks like the best source of information is really in the [source file](#) or at: web/modules/contrib/twig_tweak/src/TwigExtension.php

All the variations are listed there including the instructions to get drush to list all the blocks on your site.

```
drush ev "print_r(array_keys(Drupal::service('plugin.manager.block')->getDefinitions()));"
```

It outputs something like:

```
[37] => system_menu_block
[38] => system_messages_block
[39] => system_powered_by_block
[40] => user_login_block
[41] => views_block:comments_recent-block_1
[42] => views_block:content_recent-block_1
[43] => views_block:events-block_1
[44] => views_block:related_products_and_services-block_1
[45] => views_block:who_s_new-block_1
[46] => views_block:who_s_online-who_s_online_block
[47] => views_exposed_filter_block:news_listing_for_news_landing-page_1
[48] => local_actions_block
[49] => local_tasks_block
[50] => page_title_block
[51] => broken
```

)

Display filter form block

You can then use this to display your ajax exposed filter form block

```
{{ drupal_block('views_exposed_filter_block:news_listing_for_news_landing-page_1') }}
```

Embed view in twig template

In inside-marthe/themes/custom/dprime/templates/content/node-overview.html.twig there is a view rendered in the twig template. This requires the [twig_tweak](#) module:

```
<div class="l-sidebar-content">
  {% include '@danaprime/partials/subnav.html.twig' %}
  {{content.field_ref_sidebar}}
  {{ drupal_view('news', 'embed_page_sidebar', content.field_news_categories|render|trim) }}
</div>
```

You can, also specify additional parameters which map to contextual filters you have configured in your view.

```
 {{ drupal_view('who_s_new', 'block_1', arg_1, arg_2, arg_3) }}
```

Some tricky quotes magic

Here I am trying to create a string type="aof" so I had to escape at least one of the quotes like this \" (backslash and double quote)

```
{% set office_type = 'type=\\" ~ item.type ~ "\\" %}
```

The entire piece of debug code is reproduced below:

```
<div>
{% for filter in filter_data %}
  {% if filter.type == 'office' %}
    {% for item in filter.info %}
      type={{ filter.type }}, title = {{ item.title }}, value = {{ item.value }}, selected= {{ item.selected }}, item.type = {{ item.type }}<br>
      {% set officetype = " " %}
      {% if item.type is defined and item.type %}
        {% set office_type = 'type=\\" ~ item.type ~ "\\" %}
        Office_type:{{ office_type }}<br>
      {% endif %}
    {% endfor %}
    {% endif %}
  {% endif %}
</div>
```

The real implementation is shown below:

From txg/web/themes/custom/tgx/templates/partials/searchfilterform.html.twig

See the line below that sets office_type = ...

```
{% for item in filter.info %}
  {% set selected = " " %}
  {% if item.selected is defined and item.selected %}
    {% set selected = 'selected' %}
  {% endif %}
  {% set officetype = " " %}
  {% if item.type is defined and item.type %}
    {% set office_type = 'type=\\" ~ item.type ~ "\\" %}
  {% endif %}
  <option value="/search-news?{{ filter.type }}={{ item.value }} {{ office_type }}" {{ selected }}>{{ item.title }}</option>
{% endfor %}
```

Troubleshooting

Enable Twig debugging and disable caches

This will cause twig debugging information to be displayed in the HTML code like the following:

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'toolbar' -->
<!-- BEGIN OUTPUT from 'core/themes/stable/templates/navigation/toolbar.html.twig' -->
```

and

```

<!-- THEME DEBUG -->
<!-- THEME HOOK: 'page' -->
<!-- FILE NAME SUGGESTIONS:
  * page--teks--admin--srp--program--expectation--correlation--vote-all.html.twig
  * page--teks--admin--srp--program--expectation--correlation--852136.html.twig
  * page--teks--admin--srp--program--expectation--correlation--%.html.twig
  * page--teks--admin--srp--program--expectation--correlation.html.twig
  * page--teks--admin--srp--program--expectation--852131.html.twig
  * page--teks--admin--srp--program--expectation--%.html.twig
  * page--teks--admin--srp--program--expectation.html.twig
  * page--teks--admin--srp--program--852061.html.twig
  * page--teks--admin--srp--program--%.html.twig
  * page--teks--admin--srp--program.html.twig
  * page--teks--admin--srp.html.twig
  x page--teks--admin.html.twig
  * page--teks.html.twig
  * page.html.twig
-->
```

In `sites/default/development.services.yml` in the parameters, `twig.config`, set `debug: true`. See `core.services.yml` for lots of other items to change for development.

```

# Local development services.
#
parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    debug: true
    auto_reload: true
    cache: false

# To disable caching, you need this and a few other items
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in `settings.local.php`:

```

/***
 * Enable local development services.
*/
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in `settings.local.php`. Here all caching is disabled with:

```

$config['system.performance']['css']['preprocess'] = FALSE;
$config['system.performance']['js']['preprocess'] = FALSE;
$settings['cache']['bins']['render'] = 'cache.backend.null';
$settings['cache']['bins']['page'] = 'cache.backend.null';
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
```

Debugging - Dump a variable

When troubleshooting or trying to make sense of what is being output, use `dump`.

```
<pre>
Dump node.created.value:
{{ dump(node.created.value) }}
Dump node.changed.value:
{{ dump(node.changed.value) }}
Dump node.published_at.value:
{{ dump(node.published_at.value) }}
</pre>
```

The output might look like this. Note the published value may be null as I didn't use Drupal scheduling to publish the node:

```
Dump node.created.value:
string(10) \"1604034000\"

Dump node.changed.value:
string(10) \"1604528207\"

Dump node.published_at.value:
NULL
```

Dump taxonomy reference field

Here we dump a taxonomy reference field which is useful for debugging purposes. The pre tags format it a little nicer than if we don't have them.

```
<pre>
{{ dump(paragraph.field_ref_tax.value) }}
</pre>
```

And get ouput:

```

array(2) {
[0]=>
array(4) {
["target_id"]=>
string(2) "13"
["_attributes"]=>
array(0) {
}
["_loaded"]=>
bool(true)
["_accessCacheability"]=>
object(Drupal\Core\Cache\CacheableMetadata)#7683 (3) {
["cacheContexts":protected]=>
array(1) {
[0]=>
string(16) "user.permissions"
}
["cacheTags":protected]=>
array(0) {
}
["cacheMaxAge":protected]=>
int(-1)
}
}
[1]=>
array(4) {
["target_id"]=>
string(2) "16"
["_attributes"]=>
array(0) {
}
["_loaded"]=>
bool(true)
["_accessCacheability"]=>
object(Drupal\Core\Cache\CacheableMetadata)#7705 (3) {
["cacheContexts":protected]=>
array(1) {
[0]=>
string(16) "user.permissions"
}
["cacheTags":protected]=>
array(0) {
}
["cacheMaxAge":protected]=>
int(-1)
}
}
}
}

```

Using kint or dump to display variable in a template

With `devel` and `devel: kint` enabled, you can display variables in templates. Here we show the `content` variable from the above block template. Note. There is also a built in `dump()` function which is super useful.

```
 {{ kint(content) }}
```

You can also

```
 {{ dump(content) }}
```

And dump a value from a paragraph field. The pre tags will format the output a little more sanely.

```
<pre>  
 {{ dump(paragraph.field_ref_tax.value) }}  
</pre>
```

Or the body field:

```
 {{ kint(content['body']) }}
```

Or the tags field content['field_tags']

```
 {{ kint(content['field_tags']) }}
```

502 bad gateway error

While working on Twig changes, if you ever see a 502 bad gateway error, try commenting out the twig template code you just added and see if it displays. I know, it's not a friendly error at all!

Views error

If you ever see a 502 bad gateway error when embedding a drupal_view, delete the display and create a new one and it may just work fine.

Striptags (when twig debug info causes it to fail)

When you care about the output being affected by twig debugging, you need to use `striptags`. In this case, because I enabled twig debugging, the `content.field_landing_opinion_page_type` was not ever 'ORD'

So here I compare a field value so I have to use striptags to remove all html. I ended up using the combination of `render|striptags|trim`:

```
{% if content.field_landing_opinion_page_type|render|striptags|trim == 'ORD' %}  
 {{ drupal_block('opinion_landing', wrapper=false) }}  
{% endif %}
```

Reference

- Drupal 10 uses **Twig 3**. Drupal 9 uses Twig 2. Drupal 8 used Twig 1.
- [Theme system overview on api.drupal.org](#)
- [Twig 3 documentation](#)
- [Drupal.org Theming documentation](#)
- [Handy Twig functions you can use directly in templates - Updated Jan 2023](#)
- [Twig Tweak 3 cheat sheet Updated October 2022](#)
- [Twig Tweak 2 cheat sheet Updated May 2022](#)
- [Using attributes in templates updated March 2023](#)
- [Twig tweaks and Views has some useful notes on using twig tweak with views - Updated November 2020](#)

[Back to top](#)

[Edit this page on GitHub](#)

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



Username or email address

Password

[Forgot password?](#)

[Sign in](#)

New to GitHub? [Create an account.](#)

[Terms](#)

[Privacy](#)

[Docs](#)

[Contact GitHub Support](#)