# <https://www.drupal.org/project/content_sync>

The **content synchronization** module provides a mechanism to export single content items, or all content items, from an environment, and move them to another, effortlessly.

-----------------------------------------------------------------------------------------------------------------------------------

# In Drupal 7, hook\_init offered a hook that was fired on every page that Drupal did not cache. Drupal 8 offers this same functionality, using the [Event Subscriber](https://api.drupal.org/api/drupal/core!modules!config!tests!config_events_test!src!EventSubscriber.php/class/EventSubscriber/8.2.x) pattern as detailed on the [hook\_init change notice](https://www.drupal.org/node/2013014). These pages provide detailed examples, which walk you through the process of setting one up. [*How to Register an Event Subscriber in Drupal 8*](https://www.chapterthree.com/blog/how-to-register-event-subscriber-drupal8) also provides examples of event dispatching code.

**STATIC CLASS PHP**

Introduction: A **static class** in **PHP** is a type of **class** which is instantiated only once in a program. It must contain a **static** member (variable) or a **static** member function (method) or both. The variables and methods are accessed without the creation of an object, using the scope resolution operator(::).

# **Dynamic Permissions**

view all revisions:

title: 'View all revisions'

description: 'To view a revision, you also need permission to view the content item.'

revert all revisions:

title: 'Revert all revisions'

description: 'To revert a revision, you also need permission to edit the content item.'

delete all revisions:

title: 'Delete all revisions'

description: 'To delete a revision, you also need permission to delete the content item.'

**permission\_callbacks:**

**- \Drupal\node\NodePermissions::nodeTypePermissions**

class NodePermissions {

use StringTranslationTrait;

/\*\*

\* Returns an array of node type permissions.

\*

\* @return array

\* The node type permissions.

\* @see \Drupal\user\PermissionHandlerInterface::getPermissions()

\*/

public function **nodeTypePermissions**() {

$perms = [];

// Generate node permissions for all node types.

foreach (NodeType::loadMultiple() as $type) {

$perms += $this->buildPermissions($type);

}

return $perms;

}

# **Sub-Theme inheritance**

In the previous section, We understood how sub-theme is created along with inheritance of various properties.

The list below summarizes the inheritance properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Inherited** | **Notes** |
| CSS | Yes |  |
| JS | Yes |  |
| Templates | Yes |  |
| Screenshot | Yes |  |
| Regions | No | <https://www.drupal.org/node/2165673> |
| Features | No |  |
| Theme Settings | Yes |  |
| Core Version | No |  |
| Logo | Yes |  |
| Favicon | No |  |
| Color module support | No |  |
| Block placement and block templates | Yes, with extra steps required | <https://www.drupal.org/node/2165673> |
| Breakpoints (via themename.breakpoints.yml) | <TBD> |  |
| Libraries (via themename.libraries.yml) | Yes |  |

## PHP - What are Traits?

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviours? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the trait keyword:

## In Drupal 8

## If we look inside controllerbase class

## We will find

abstract class ControllerBase implements ContainerInjectionInterface {

use StringTranslationTrait;

use LinkGeneratorTrait;

use UrlGeneratorTrait;

...more code goes here...

}

The first thing I notice are these three lines:

use StringTranslationTrait;

use LinkGeneratorTrait;

use UrlGeneratorTrait;

## that’s where the traits are used in Drupal 8

## <https://drupalize.me/blog/201503/dependency-injection-traits-drupal-8>

# Explain Encapsulation in PHP.

[PHP](https://www.tutorialspoint.com/questions/category/PHP)[Object Oriented Programming](https://www.tutorialspoint.com/questions/category/Object-Oriented-Programming)[Programming](https://www.tutorialspoint.com/questions/category/Programming)

Object Oriented Programming is a software approach added to PHP5, which helps in building the composite application in an easy way. Some of the OOP concepts added into the PHP5 are an abstraction, interface, static method, and static class, etc...

In this article, we will learn Encapsulation and it's implementation through a few examples.

The wrapping up of data and methods into a single unit (called class) is known as encapsulation. Encapsulation is a protection mechanism for the data members and methods present inside the class. In the encapsulation technique, we are restricting the data members from access to outside world end-user.

In PHP, encapsulation utilized to make the code more secure and robust. Using encapsulation, we are hiding the real implementation of data from the user and also does not allow anyone to manipulate data members except by calling the desired operation.

## Example

Let' understand this through an example.

<?php

   class ATM {

      private $custid;

      private $atmpin;

      public function PinChange($custid,$atmpin) {

               ---------perform tasks-----

               }

      public function CheckBalance($custid,$atmpin){

               ---------perform tasks-----

               }

      public function miniStatement($custid) {

               ---------perform tasks-----

               }

      }

   $obj = new ATM();

   $obj ->CheckBalance(10005285637,1\*\*3);

?>

## PHP - What are Abstract Classes and Methods?

## Abstract classes are the classes in which at least one method is abstract. Unlike C++ abstract classes in PHP are declared with the help of abstract keyword. Use of abstract classes are that all base classes implementing this class should give implementation of all abstract methods declared in parent class.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the abstract keyword:

### **Syntax**

<?php  
abstract class ParentClass {  
  abstract public function someMethod1();  
  abstract public function someMethod2($name, $color);  
  abstract public function someMethod3() : string;  
}  
?>

**Interface**

It specifies the lists of all such methods that a class must implement. Use the keyword *Interface* to implement interface same as a class. It can extend an interface using the extends operator. **All the methods in Interface are abstract methods** and can have their own constants. There is a concrete class concept which is a class that implements an interface which must implement all methods having the same names and signatures.  
All the methods in the interface must have a public access level.

**Syntax:**

filter\_none

brightness\_4

|  |
| --- |
| <?php  // A sample interface in PHP  interface MyInterface  {   // function...  } |

No two interface can be implemented by a particular class having same method name and signatures because it give error. Also helps in multiple inheritance because a class can implement more than one interface whereas it can extend only one class. Implementations can be changed without affecting the caller of the interface.

**Example:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| <?php  // PHP program to demonstrate working  // of interface.  interface MyInterface{        public function examplemethod1();      public function examplemethod2();    }    class MyClass implements MyInterface{        public function examplemethod1(){          echo "ExampleMethod1 Called" . "\n";      }        public function examplemethod2(){          echo "ExampleMethod2 Called". "\n";      }  }    $ob = new MyClass;  $ob->examplemethod1();  $ob->examplemethod2();    ?> |

**Output:**

ExampleMethod1 Called

ExampleMethod2 Called

# **Polymorphism in PHP**

In this tutorial, we are going to learn about **Polymorphism** (Greek for "many forms") a naming convention that can help us write code which is much more coherent and easy to use. According to the **Polymorphism** principle, methods in different classes that do similar things should have the same name.

*According to the Polymorphism principle, methods in different classes that do similar things should have the same name.*

## What is Polymorphism?

Polymorphism is a long word for a very simple concept.

*Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.*

## OOPS Concepts in Drupal 8

## <https://api.drupal.org/api/drupal/developer%21topics%21oop.html/4.7.x>

### Abstraction – Drupal Hooks

### Encapsulation - Since Drupal code is based around functions, which share a single namespace, this namespace is subdivided by the use of prefixes. By following this simple convention, each module can declare its own functions and variables without the worry of conflict with others.

### Polymorphism - Nodes are polymorphic in the classical sense.

### Inheritance - Modules and themes can define whatever functions they please. However, they can both be thought to inherit their behavior from an abstract base class.

## Why Twig? It’s more secure.

In Drupal 7, user-submitted text needed to be sanitized with check\_plain() in order to prevent the most common web vulnerability, Cross Site Scripting (XSS). If a themer forgot to sanitize their output there would be a security hole. Autoescaping was [recently accepted](https://www.drupal.org/node/1825952) into Drupal 8, which removed this concern all together. PHP functions will also be stripped from templates and this goes in line with separation of concerns. Instead, you’ll have concise, clean markup such as:

“The most powerful part of Twig is template inheritance” – *[Sensiolabs](http://twig.sensiolabs.org/doc/templates.html" \t "_blank)*

Drupal 8 + Twig integration eliminates the need for copying and pasting base or parent theme template files into your custom templates. Similarly to how Sass and Less simplifies your CSS workflow with the @extend directive, Twig will drastically cut down the amount of template files and code you need to organize your theme.

**{% extends "themes/sub\_bartik/templates/node.html.twig" %}**

This is similar to PHP’s “include function” that allows you to create dynamic, hookable templates. However, there’s a huge leap forward in templating that I’m about to show you: Twig blocks (not to be confused with Drupal blocks).

<https://www.drupal.org/project/ape>

# **Advanced Page Expiration**

## Advanced control of your cache-control header

Advanced Page Expiration addresses the use cases of needing certain pages on the site to expire faster than other pages. For example, using this module all pages on the site could be cached for 24 hours except for the homepage which could be cached for five minutes.

Advanced Page Expiration allows for better control of the Cache-Control header when using external caching servers such as Varnish. Every site with an external caching server that recognizes the use of a cache-control header to set TTL for cached pages such as Varnish and nginx should consider this module.

In Drupal this is controlled by the poorly named "**Expiration of cached pages**" dropdown on the performance configuration page. What Drupal isn't clear about is that this dropdown sets a cache-control header for Drupal when it responds to a request. This header is what tells caching servers like Varnish how long the URL should be kept in the cache.

# **Varnish: Bypassing the cache**

<https://support.acquia.com/hc/en-us/articles/360004352373-Varnish-Bypassing-the-cache>

**Syntax to define local task in Drupal 8:**

### if module name is "resume", the file will be ***resume.links.task.yml*** & placed in the root directory.

|  |
| --- |
| resume.tab\_1: |
|  | route\_name : resume.form |
|  | title: Resume |
|  | base\_route: resume.form |
|  | weight: 10 |
|  |  |
|  | resume.tab\_2: |
|  | route\_name: work.form |
|  | title: Work |
|  | base\_route: resume.form |
|  | weight: 20 |

### **DEFINITION OF DERIVATIVE CLASS DEFINED IN BLOCK ANNOTATION**

We added the derivative class to the annotation and we retrieve the block id to pass it on to a function that retrieves content based on the given argument.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | <?php    /\*\*  \* @file  \* Contains \Drupal\mymodule\Plugin\Derivative\MyModuleBlock.  \*/    namespace Drupal\mymodule\Plugin\Derivative;  use Drupal\Core\Plugin\Discovery\ContainerDerivativeInterface;    /\*\*  \* Provides block plugin definitions for mymodule blocks.  \*  \* @see \Drupal\mymodule\Plugin\Block\MyModuleBlock  \*/  **class MyModuleBlock extends DerivativeBase implements ContainerDerivativeInterface** {    /\*\*     \* {@inheritdoc}     \*/    public function **getDerivativeDefinitions**(array $base\_plugin\_definition) {      $myblocks = array(        'mymodule\_block\_first' => t('MyModule Block: First'),        'mymodule\_block\_second' => t('MyModule Block: Second'),      );      foreach ($myblocks as $block\_id => $block\_label) {        $this->derivatives[$block\_id] = $base\_plugin\_definition;        $this->derivatives[$block\_id]['admin\_label'] = $block\_label;        $this->derivatives[$block\_id]['cache'] = DRUPAL\_NO\_CACHE;      }      return $this->derivatives;    }  } |
|  |  |

# **Create custom twig templates for custom module**

**Step #1: Define hook\_theme in .module file**

**Define template in hook\_theme**  
Create a [module].module file if it doesn't already exist, and add code that defines each of your twig templates. The key of each item in the array is what you will need to call the template later. Do not use dashes in the file name.

function test\_twig\_theme($existing, $type, $theme, $path) {

return [

'my\_template' => [

'variables' => ['test\_var' => NULL],

],

];

}

See the documentation for [hook\_theme()](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Render%21theme.api.php/function/hook_theme/8).

**Step #2: Call the Template**

**Define template used for your own controller by #theme**  
In the place where you are returning your render array (whether from a controller method that is called from your router yml file, or wherever), make a call to your twig template. Below is an example from a testing module that is called from the routing yml file in the module. (need more info on this part)

/\*\*

\* @file

\* Contains \Drupal\test\_twig\Controller\TestTwigController.

\*/

namespace Drupal\test\_twig\Controller;

use Drupal\Core\Controller\ControllerBase;

class TestTwigController extends ControllerBase {

public function content() {

return [

'#theme' => 'my\_template',

'#test\_var' => $this->t('Test Value'),

];

}

}

### **Disable cache for a particular route?**

**mymodule.myroute:  
 path: '/mymodule/mypage'  
 defaults:  
 \_controller: '\Drupal\mymodule\Controller\Pages::mypage'  
 \_title: 'No cache page'  
 requirements:  
 \_access: 'TRUE'  
 options:  
 no\_cache: 'TRUE'**

**Disable cache for a particular block?**

Disable cache for a particular block  
class MYCUSTOMBLOCK extends BlockBase {

*/\*\**  
*\* {@inheritdoc}*  
*\*/*  
**public** function **build**() {  
 **return** array(  
 '#markup' => ""  
 );  
}

*/\*\**  
*\* {@inheritdoc}*  
*\*/*  
**public** function **getCacheMaxAge**() {  
 **return** 0;  
}

}

### **what are events in drupal 8?**

Events in Drupal 8 allow various system components to interact and communicate with one another while remaining independent, or decoupled. The event system is built on the [Symfony event dispatcher component](https://symfony.com/doc/current/components/event_dispatcher.html), and is an implementation of the [Mediator design pattern](https://en.wikipedia.org/wiki/Mediator_pattern).

## How to define your own Event in Drupal 8?

|  |
| --- |
| <?php |
|  |  |
|  | namespace Drupal\example\_events; |
|  |  |
|  | **use Symfony\Component\EventDispatcher\Event;** |
|  |  |
|  | **class ExampleEvent extends Event {** |
|  |  |
|  | const SUBMIT = 'event.submit'; |
|  | protected $referenceID; |
|  |  |
|  | public function \_\_construct($referenceID) |
|  | { |
|  | $this->referenceID = $referenceID; |
|  | } |
|  |  |
|  | public function getReferenceID() |
|  | { |
|  | return $this->referenceID; |
|  | } |
|  |  |
|  | public function myEventDescription() { |
|  | return "This is as an example event"; |
|  | } |
|  | } |

## How to dispatch the event in Drupal 8?

|  |
| --- |
| <?php |
|  | // Following is the example for How to dispatch an event in Drupal 8? |
|  |  |
|  | // Use the namespace of the ExampleEvent class |
|  | use Drupal\example\_events\ExampleEvent; |
|  |  |
|  | // load the Symfony event dispatcher object through services |
|  | $dispatcher = \Drupal::service('event\_dispatcher'); |
|  |  |
|  | // creating our event class object. |
|  | $event = new ExampleEvent($form\_state->getValue('name')); |
|  |  |
|  | // dispatching the event through the ‘dispatch’ method, |
|  | // passing event name and event object ‘$event’ as parameters. |
|  | $dispatcher->dispatch(ExampleEvent::SUBMIT, $event); |

**$dispatcher = \Drupal::service('event\_dispatcher');**

**$event = new ExampleEvent($form\_state->getValue('name'));**

**$dispatcher->dispatch(ExampleEvent::SUBMIT, $event);**

## How to subscribe the events in Drupal 8?

|  |
| --- |
|  |
| **class ExampleEventSubScriber implements EventSubscriberInterface {** |
|  |

### Define your event Subscriber class that implements Symfony EventSubscriberInterface.

We should defining the Event Subscriber class under ‘**ModuleRoot/src/EventSubscriber/**’ directory, In our example, path will be,

“**example\_events/src/EventSubscriber/ExampleEventSubScriber.php**”

|  |  |
| --- | --- |
|  | <?php |
|  |  |
|  | /\*\* |
|  | \* @file |
|  | \* Contains \Drupal\example\_events\ExampleEventSubScriber. |
|  | \*/ |
|  |  |
|  | namespace Drupal\example\_events\EventSubscriber; |
|  |  |
|  | use Drupal\Core\Config\ConfigCrudEvent; |
|  | use Drupal\Core\Config\ConfigEvents; |
|  | use Drupal\example\_events\ExampleEvent; |
|  | use Symfony\Component\EventDispatcher\EventSubscriberInterface; |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Class ExampleEventSubScriber. |
|  | \* |
|  | \* @package Drupal\example\_events |
|  | \*/ |
|  | class ExampleEventSubScriber implements EventSubscriberInterface { |
|  |  |
|  | /\*\* |
|  | \* {@inheritdoc} |
|  | \*/ |
|  | public static function getSubscribedEvents() { |
|  | $events[ConfigEvents::SAVE][] = array('onSavingConfig', 800); |
|  | $events[ExampleEvent::SUBMIT][] = array('doSomeAction', 800); |
|  | return $events; |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Subscriber Callback for the event. |
|  | \* @param ExampleEvent $event |
|  | \*/ |
|  | public function doSomeAction(ExampleEvent $event) { |
|  | drupal\_set\_message("The Example Event has been subscribed, which has bee dispatched on submit of the form with " . $event->getReferenceID() . " as Reference"); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Subscriber Callback for the event. |
|  | \* @param ConfigCrudEvent $event |
|  | \*/ |
|  | public function onSavingConfig(ConfigCrudEvent $event) { |
|  | drupal\_set\_message("You have saved a configuration of " . $event->getConfig()->getName()); |
|  | } |
|  | } |

### Tag your Event Subscriber Class with event\_subscriber

In your **module.services.yml** tag your event subscriber class as ‘**event\_subscriber**’. Just like our event class. Tags indicate these service should be registered or for specific purpose, or that it belongs to a category. For more details on tags refer: [Symfony documentation on service tags.](http://symfony.com/doc/current/book/service_container.html" \l "book-service-container-tags)

|  |  |
| --- | --- |
|  | services: |
|  | example\_events.event\_subscriber\_example: |
|  | class: Drupal\example\_events\EventSubscriber\ExampleEventSubScriber |
|  | tags: |
|  | - { name: 'event\_subscriber' } |

### **Defining a library**

# fluffiness.libraries.yml

cuddly-slider:

version: 1.x

css:

theme:

css/cuddly-slider.css: {}

js:

js/cuddly-slider.js: {}

#### Including Jquery in your Library

# fluffiness.libraries.yml

cuddly-slider:

version: 1.x

css:

theme:

css/cuddly-slider.css: {}

js:

js/cuddly-slider.js: {}

dependencies:

- core/jquery

#### Declaring dependencies

# fluffiness.libraries.yml

new\_library:

js:

js/new\_libary.js: {}

dependencies:

- core/jquery

- my\_module/my\_library

- my\_theme/my\_library

### **Attaching a library to all pages**

# fluffiness.libraries.yml (multiple libraries can be added to a libraries.yml file, these would appear below the cuddly-slider libraries added earlier)

global-styling:

version: 1.x

css:

theme:

css/layout.css: {}

css/style.css: {}

css/colors.css: {}

global-scripts:

version: 1.x

js:

js/navmenu.js: {}

# 

To be available everywhere in the theme, the global-styling/global-scripts libraries must then be added to your theme's info.yml (in this case fluffiness.info.yml)

#fluffiness.info.yml

name: Fluffiness

type: theme

description: 'A cuddly theme that offers extra fluffiness.'

core: 8.x

# by adding global-styling and global-scripts here, the css/js files in the library become

# available to every page presented by the theme

libraries:

- fluffiness/global-styling

- fluffiness/global-scripts

base theme: classy

regions:

header: Header

content: Content

sidebar\_first: 'Sidebar first'

footer: Footer

For instance, if you want to attach JavaScript to the maintenance page, the "HOOK" part is "maintenance\_page", and your function would look like this:

function fluffiness\_preprocess\_maintenance\_page(&$variables) {

$variables['#attached']['library'][] = 'fluffiness/cuddly-slider';

}

ways to use libraries-override to remove or replace CSS or Javascript assets or entire libraries your theme has inherited from modules or themes.

**libraries-override:**

# Replace an entire library.

core/drupal.collapse: mytheme/collapse

# Replace an asset with another.

subtheme/library:

css:

theme:

css/layout.css: css/my-layout.css

# Replace an override asset from stable:

contextual/drupal.contextual-toolbar:

css:

component:

/core/themes/stable/css/contextual/contextual.toolbar.css: css/contextual.toolbar.css

# Replace a core module JavaScript asset.

toolbar/toolbar:

js:

js/views/BodyVisualView.js: js/views/BodyVisualView.js

# Remove an asset.

drupal/dialog:

css:

theme:

dialog.theme.css: false

# Remove an entire library.

core/modernizr: false

### **libraries-extend**

libraries-extend provides a way for themes to alter the assets of a library by adding in additional theme-dependent library assets whenever a library is attached.  
libraries-extend are specified by extending a library with any number of other libraries.

# Extend drupal.user: add assets from classy's user libraries.

libraries-extend:

core/drupal.user:

- classy/user1

- classy/user2

 By default, all JS assets are now loaded in the footer. JS for critical UI elements that cannot be shown unless their corresponding JS has run can be loaded in the header if needed like so:

js-header:

header: true

js:

header.js: {}

js-footer:

js:

footer.js: {}

# **Add JS**

add library in twig:

{{ attach\_library('core/drupal.dialog.ajax') }}

Twig

Copy

add library in PHP:

$variables['#attached']['library'][] = 'core/drupal.dialog.ajax';

# **Hide region if empty in Drupal 8**

{% if page.footer|render|trim %}

# **Twig**

# **Copy**

or use can use this example:

{% if content.footer|render|trim is not empty %}

# **Insert data in custom Table**

in this post, I'll show you how to create a custom table to store data in the database.

to create your custom table you need minimum the **hook\_schema()** method in the **mymodule.install** file. also I use **hook\_install()**, which is called after the **hook\_schema()** to store some examples in our mymodule Table.

**mymodule.install**

**<?php**

/\*\*

\* hook\_install()

\*/

function mymodule\_install() {

$values = [

[

'name' => 'Jhon',

'age' => 30,

'uid' => 1,

],

[

'name' => 'Jan',

'age' => 28,

'uid' => 1,

],

];

$database = \Drupal::database();

$query = $database->insert('mymodule')->fields(['name', 'age', 'uid']);

foreach ($values as $developer) {

$query->values($developer);

}

$query->execute();

}

/\*\*

\* hook\_schema()

\*/

function mymodule\_schema() {

$schema['mymodule'] = array(

'description' => 'Store developers',

'fields' => array(

'did' => array(

'type' => 'serial',

'not null' => TRUE,

'description' => 'Primary Key: Unique developer ID.',

),

'uid' => array(

'type' => 'int',

'not null' => TRUE,

'default' => 0,

'description' => "Creator user's {users}.uid",

),

'name' => array(

'type' => 'varchar',

'length' => 255,

'not null' => TRUE,

'default' => '',

'description' => 'Name of the developer.',

),

'age' => array(

'type' => 'int',

'not null' => TRUE,

'default' => 0,

'size' => 'tiny',

'description' => 'The age of the developer in years.',

),

),

'primary key' => array('did'),

'indexes' => array(

'name' => array('name'),

'age' => array('age'),

),

);

return $schema;

}

# **Drupal 8: How to override page title**

# **use $variables['title']  in function THEMENAME\_preprocess\_page\_title(&$variables) { } to set the page title.**

/\*\*

\* @param $variables

\*/

function mytheme\_preprocess\_page\_title(&$variables) {

if ($node = \Drupal::routeMatch()->getParameter('node')) {

switch ($node->getType()){

case "article":

$variables['title'] = t('News');

break;

case "page":

if ( $node->id() == "3"){

$variables['title'] = t('Page Title');

}

break;

}

}

}

# 

# **Arguments in Controller**

example.user\_form:

path: '/example/form/{user}'

defaults:

\_form: '\Drupal\example\Form\ExampleForm'

requirements:

\_permission: 'access content'

namespace Drupal\example\Form;

use Drupal\Core\Form\FormBase;

use Drupal\Core\Session\AccountInterface;

class ExampleForm extends FormBase {

public function buildForm(array $form, FormStateInterface $form\_state, AccountInterface $user = NULL) {

// Do something with $user in the form

}

}

# **Passing a Default Value to a route**

issue.report\_form:

path: '/report/{issue\_type}'

defaults:

\_controller: '\Drupal\issue\Controller\IssueController::report'

issue\_type: 'support-request'

requirements:

\_permission: 'report issue'

# **Custom controller with JSON response in Drupal 8**

[**https://codimth.com/blog/custom-controller-json-response-drupal-8**](https://codimth.com/blog/custom-controller-json-response-drupal-8)

**create mymodule.routing.yml**

codimth.json\_api\_articles:

path: '/api/articles'

defaults:

\_controller: 'Drupal\mymodule\Controller\JsonApiArticlesController::index'

\_title: 'Codimth JSON api'

methods: [GET]

requirements:

\_access: 'TRUE'

# **YAML**

# **Copy**

**create src/Controller/JsonApiArticlesController.php**

**getData():**will return article nodes from database.

**<?php**

namespace Drupal\mymodule\Controller;

use Symfony\Component\HttpFoundation\JsonResponse;

/\*\*

\* Class JsonApiArticlesController

\* @package Drupal\mymodule\Controller

\*/

class JsonApiArticlesController {

/\*\*

\* @return JsonResponse

\*/

public function index() {

return new JsonResponse([ 'data' => $this->getData(), 'method' => 'GET', 'status'=> 200]);

}

/\*\*

\* @return array

\*/

public function getData() {

$result=[];

$query = \Drupal::entityQuery('node')

->condition('type', 'article')

->sort('title', 'DESC');

$nodes\_ids = $query->execute();

if ($nodes\_ids) {

foreach ($nodes\_ids as $node\_id) {

$node = \Drupal\node\Entity\Node::load($node\_id);

$result[] = [

"id" => $node->id(),

"title" => $node->getTitle(),

];

}

}

return $result;

}

}

# **PHP**

# **Copy**

**/api/articles** will return result

# **BigPipe**

<https://www.drupal.org/project/big_pipe>

Sends pages in a way that allows browsers to show them much faster. First sends the cacheable parts of the page, then the dynamic/uncacheable parts. Uses the [BigPipe technique](https://www.facebook.com/notes/facebook-engineering/bigpipe-pipelining-web-pages-for-high-performance/389414033919).

**What are plugins in drupal8?**

<https://www.valuebound.com/resources/blog/drupal-8-extending-module-using-plugin-manager>

**Drupal 8 introduces plugins. Plugins power many items in Drupal, such as blocks, field types, field formatters, and many more. Plugins and plugin types are provided by modules. They provide a swappable and specific functionality. Breakpoints, as discussed in Chapter 5, Front End for the Win, are plugins. In this chapter, we will discuss how plugins work in Drupal 8 and show you how to create blocks, fields, and custom plugin types.**

**what is plugin manager in Drupal 8?**

The **plugin manager** is the central controlling class that defines how the **plugins** of a particular type will be discovered and instantiated. This class is called directly in any module wishing to invoke a **plugin** type.

PLUGIN MANAGER ARE DEFINED AS A SERVICE.

**Difference between private and protected in php?**

**Private**: Method or property with **private** visibility can only be accessible inside the class. You can not access **private** method or variable from outside of your class.

**Protected**: Method or variable with **protected** visibility can only be access **in the** child class. **Protected** will be used **in the** process of inheritance.

**CREATE A MENU LINK IN DRUPAL 8**

Step 1: Create a new directory in the modules directory called *hello*.

Step 2: In the hello directory, create a file called *hello.info.yml* and add the following code to that file:

name: Hello

description: An experimental module to build our first Drupal 8 module

package: Custom

type: module

version: 1.0

core: 8.x

Step 3: In the hello directory, create a file for the route called *hello.routing.yml* and add the following code to that file:

hello.content:

path: '/hello/{name}'

defaults:

\_controller: 'Drupal\hello\Controller\HelloController::content'

\_title: 'Hello world'

name: 'there'

requirements:

\_permission: 'access content'

Step 4: In the hello directory, create a new directory called src and inside that create a directory called Controller. In the Controller directory, create a file for the controller called *HelloController.php* and add the following code to it. Add open PHP tags to the top of the file (I currently can't add this to the code snippet as it breaks the code highlighter).

<?php

/\*\*

\* @file

\* Contains \Drupal\hello\Controller\HelloController.

\*/

namespace Drupal\hello\Controller;

use Drupal\Core\Controller\ControllerBase;

class HelloController extends ControllerBase {

public function content($name) {

return array(

'#type' => 'markup',

'#markup' => $this->t('Hello @name', array('@name' => $name)),

);

}

}

Step 5: In the hello directory, create a file for the menu links called ***hello.links.menu.yml*** and add the following code to it:

hello.demo:

title: 'Hello'

description: 'Hello page'

parent: main

menu\_name: main

route\_name: hello.content

Step 6: Optionally change the menu link and nest it under the Home menu link:

hello.demo:

title: 'Hello'

description: 'Hello page'

parent: standard.front\_page

menu\_name: main

route\_name: hello.content

Step 7: Clear the cache

**SERVICES IN DRUPAL 8?**

In **Drupal 8** speak, a service is any object managed by the **services** container. **Drupal 8** introduces the concept of **services** to decouple reusable functionality and makes these **services** pluggable and replaceable by registering them with a service container.  
<https://cipix.nl/understanding-drupal-8-part-2-service-container>

## Tagged services

Tags are used to load certain 'tagged' services. In practice, these are sometimes used in a 'hooky' way in Drupal. In the example below (node.services.yml) the node module adds an access checker for the 'add node' page that is based on the access\_check services pattern, which is used after routing to find out if access should be granted:

services:  
  ...  
  access\_check.node.add:  
    class: Drupal\node\Access\NodeAddAccessCheck  
    arguments: ['@plugin.manager.entity']  
    tags:  
      - { name: access\_check }  
        ...

## Compiler passes

How does Drupal understand these tags and know what to do with them? Well, Symfony has yet another method of dynamically configuring the service container: **compiler passes**. The service container is actually compiled directly after building it from the static configuration. At several points during this phase, it allows objects implementing the **CompilerPassInterface** to modify the configuration. In Drupal, the **CoreServiceProvider** registers some important compiler passes, such as the RegisterAccessChecksPass, which attempts to find all services tagged with access\_check (see previous example) and adds it to the AccessManager (by using the addMethodCall container service directive). When, in the routing phase, the AccessManager service is asked to check for access, it will amongst others, check the NodeAddAccessCheck. There are several other compiler passes in the Drupal core, such as for converting route parameters to objects and translating strings. In practice you will sometimes need to use these tagged services in your module to add custom access checks and convert path parameters!

## Swapping services

The service container makes flexible service configuration possible. Drupal 8 makes use of service arguments to alter the way that Symfony is functioning for its own purposes. For example, Drupal needs a different url routing mechanism than Symfony. It does so by providing an alternative 'router' service (Symfony\Cmf\Component\Routing\ChainRouter) compared to the one used by Symfony (Symfony\Bundle\FrameworkBundle\Routing\Router). This can be done without having to change any line of code in the router\_listener service (Symfony\Component\HttpKernel\EventListener\RouterListener) itself! This is possible because both routers implement the RequestMatcherInterface, which is the one requirement for the RouterListener's first argument.

## Conclusion

In this part, we have learned about the Service Container component in Drupal 8. You now have a better understanding on how Drupal 8 injects its own services instead of Symfony2 without having to change a line of code. This allows Symfony2 to be updated easily by the Drupal core maintainers. When learning about Drupal 8, you will often have to consult the core.services.yml file to find out which services are being used.

In the next part of this article we’ll have a detailed look at the [general flow of control in Drupal 8 and routing](https://cipix.nl/understanding-drupal-8-part-3-routing).

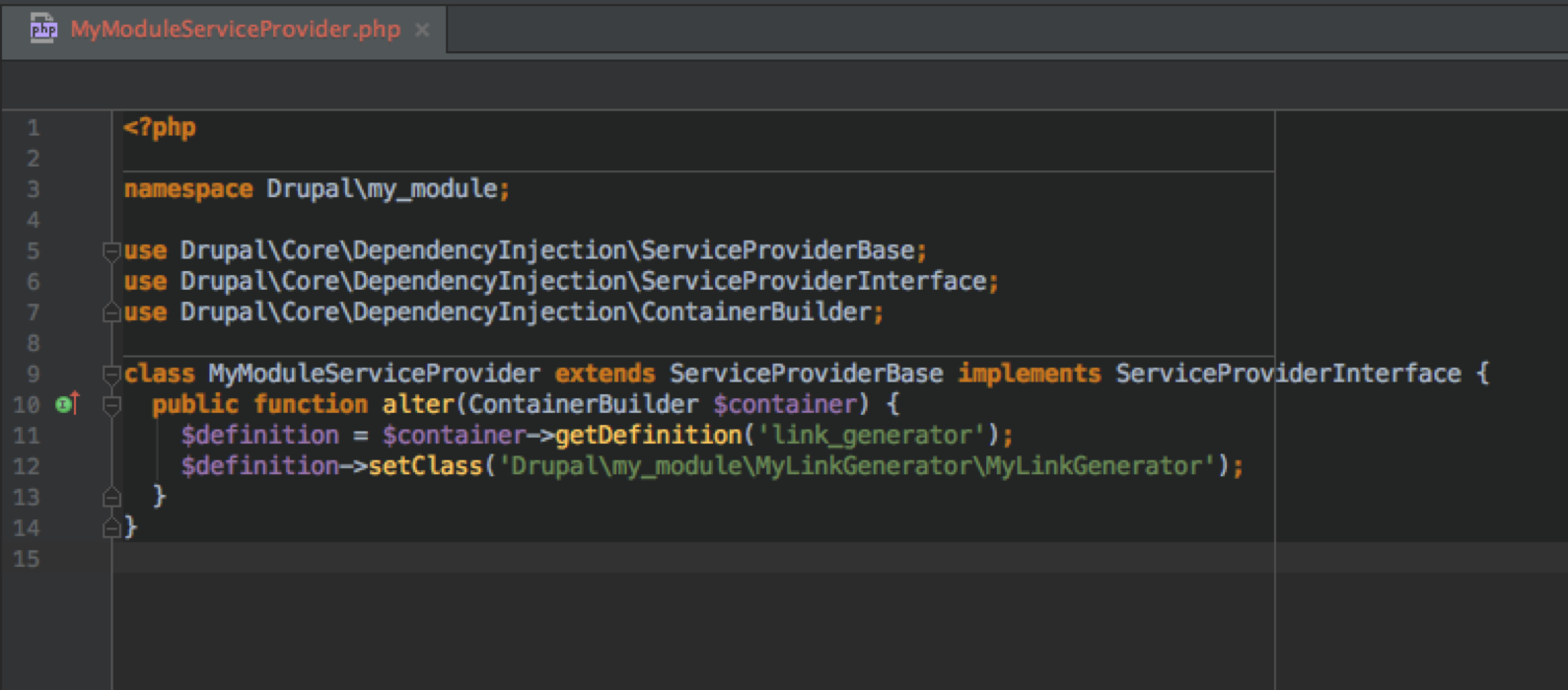
**Name some of drupal core services?**

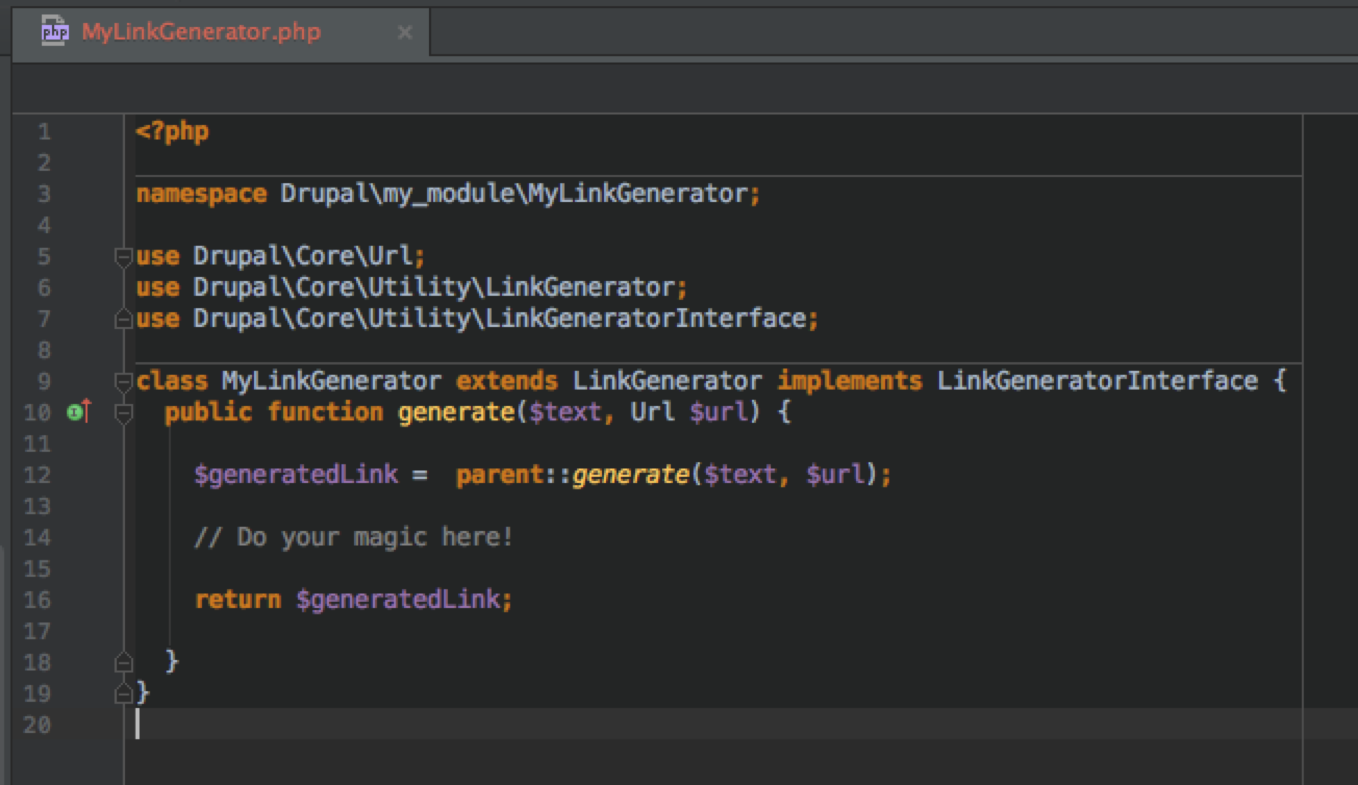
<https://www.drupal.org/docs/8/api/services-and-dependency-injection/services-and-dependency-injection-in-drupal-8>

**How to override drupal core services?**

<https://www.bounteous.com/insights/2017/04/19/drupal-how-override-core-drupal-8-service/>

**YourClassProvider extends ServiceProviderBase and implements ServiceProviderInterface**





**Expose CUSTOM TABLE TO VIEWS?**

<https://www.drupal.org/project/view_custom_table>

View custom table module provide you functionality to integrate your custom table data to views, and access all it's column in views. This module use hook\_view\_data to add custom tables in views. this module provides you following functionalities.

**Cache tags FTW!**

<https://gorannikolovski.com/blog/block-caching-examples>

Cache Tags are one of the key improvements of Drupal 8. They allow Drupal to know which entities are used during a page view. This means that Drupal saves the cache tags along with the data it caches. ***So if Drupal needs to clear the caches because one entity is updated (let’s assume node with ID 77), instead of* clearing all caches**, it can easily check ***which caches actually used node 77 and clear only those***. (You can learn more about cache tags on [drupal.org](https://www.drupal.org/docs/8/api/cache-api/cache-tags)).

A cache tag is a string.

Cache tags are passed around in sets (order doesn't matter) of strings, so they are typehinted to string[]. They're sets because a single cache item can depend on (be invalidated by) many cache tags.

By convention, they are of the form thing:identifier — and when there's no concept of multiple instances of a thing, it is of the form thing. The only rule is that it cannot contain spaces.

**what preprocess function can do apart from variables ovverride?**

You can affect the output of certain HTML via preprocess functions. For example, if you wanted to add a class to a menu and preferred to do this at the PHP level you can. This is a good way to alter theme-specific markup, but if you want to make theme-independent markup it is better to code a custom module.

To work with preprocess functions:

1. Create or edit a file in your theme directory called mytheme.theme
2. Create a function such as mytheme\_preprocess\_HOOK where HOOK refers to the item you wish to affect \*
3. Write your changes and save
4. Rebuild the cache so your changes are available (if you have drush installed, drush cr on the command line)

\* HOOK names follow twig template suggestions. To create a hook for page.html.twig you create mytheme\_preprocess\_page. To create a hook for node--article.html.twig you create mytheme\_preprocess\_node\_\_article (replacing dashes with underscores). To discover hook names, see [Locating Template Files with Debugging](https://www.drupal.org/node/2358785).

Let's assume we wanted to add a class of my-menu to all of the menus on your site. Assuming your theme is called "mytheme" you would write the following function:

/\*\*

\* Implements hook\_preprocess\_HOOK() for menu.html.twig.

\*/

function mytheme\_preprocess\_menu(&$variables) {

// If there is not an existing class array, create an empty array.

if (!isset($variables['attributes']['class'])) {

$variables['attributes']['class'] = [];

}

// Merge with any classes that may have been set by other hook\_preprocess\_menu invocations

$variables['attributes']['class'] = array\_merge($variables['attributes']['class'], ['my-menu']);

}

Differences from Drupal 7

* There is no longer a template.php file. That file has been replaced by mytheme.theme. However, it still functions in much the same way allowing for hooks to modify output.

**HOW TO CREATE OWN PROFILE IN DRUPAL 8?**

### **Creating the file structure**

<https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile>

Your installation profile will reside in its own profilename directory in the /profiles directory of a Drupal 8 site.

All installation profiles must have a [profilename.info.yml](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile" \l "info) file. They may also have:

1. [profilename.install file](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#install)
2. [profilename.profile](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#profile)
3. [config folder](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#config)
4. [translations folder](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#translations)
5. *profilename*.info.yml file should look similar to this:
6. name: Profile Name
7. type: profile
8. description: 'Description of your profile.'
9. core: 8.x

**How to switch themes programmatically?**

<https://jimconte.com/blog/web/dynamic-theme-switching-in-drupal-8>

Drupal 8's **ThemeNegotiatorInterface** provides a way for developers to create their own management class that decides which theme should go with which content.

## Implement The Interface

To start, [take a look at the documentation](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Theme!ThemeNegotiatorInterface.php/interface/ThemeNegotiatorInterface), and create a class that implements the interface.

**<?php**

/\*\*

\* @file

\* Contains \Drupal\jcmodule\Theme\ThemeNegotiator

\*/

namespace Drupal\jcmodule\Theme;

use Drupal\Core\Routing\RouteMatchInterface;

use Drupal\Core\Theme\ThemeNegotiatorInterface;

class ThemeNegotiator implements ThemeNegotiatorInterface {

View source

ThemeNegotiatorInterface requires two methods to be implemented

// Whether this theme negotiator should be used to set the theme.

function applies(RouteMatchInterface $route\_match)

// Determines the active theme for the request.

function determineActiveTheme(RouteMatchInterface $route\_match)

View source

If applies return true, determineActiveTheme is called and either the theme is chosen or null is returned

To keep my class easy to maintain, I created my own common negotiateRoute function that satisfies the requirements for both interface functions.. If my function returns a theme name, that theme is used.

Here is my common class function

/\*\*

\* Function that does all of the work

\* @param RouteMatchInterface $route\_match

\* @return null|string

\*/

private function negotiateRoute(RouteMatchInterface $route\_match)

{

if ($route\_match->getRouteName() == 'user.login')

{

return 'seven';

}

elseif ($route\_match->getRouteName() == 'some.other.route')

{

return 'some\_other\_theme';

}

return null;

}

View source

## Symfony Service Requirement

For Drupal to find your implementation, you need to set up your class as a [Symfony service](http://symfony.com/doc/2.8/service_container.html" \t "_blank) and tag it.

# Module services file jcmodule.services.yml

services:

jcmodule.theme.negotiator:

class: Drupal\jcmodule\Theme\ThemeNegotiator

tags:

- { name: theme\_negotiator, priority: 1000 }

View source

* The tag name theme\_negotiator tells Drupal that this is a class that implements ThemeNegotiatorInterface, and should be used to determine the theme in the current request.
* In my example, routes are checked to determine the theme, but you can add whatever logic you may need. This is where you can get creative. Switching the theme based on a user's role can be done within this context:
* $userRolesArray = \Drupal::currentUser()->getRoles();
* if (in\_array("administrator", $userRolesArray))
* {
* return 'seven';
* }

**Load template of a page based on field value of the node?**

We add theme settings in theme-settings.php. If you don’t already have it, add it in the root of your theme, same place as mytheme.info.yml

We define variables in mytheme.theme

Conditional statements in Twig:

{% if the\_thing %}  
 // do this  
{% endif %}

{% if mytheme\_contact\_title %}  
<h2>{{ mytheme\_contact\_title }}</h2>  
{% endif %}

{% if mytheme\_contact\_title or mytheme\_contact\_address or mytheme\_contact\_email or mytheme\_contact\_phone or mytheme\_contact\_phone %} // that whole contact block{% endif %}

Provide a custom template/hook for your custom module.

**Drupal 8**

**drupalSettings.path.baseUrl**

**Drupal 8 method**

In the lotus.module file:

/\*\*

\* Implements hook\_theme() to add the template definition.

\*\*/

function lotus\_theme($existing, $type, $theme, $path) {

return array(

'lotus\_template' => array(

'variables' => array('test\_var' => NULL),

),

);

}

In the LotusController.php file:

//Calling from the Controller

/\*\*

\* @file

\* Contains \Drupal\lotus\Controller\LotusController.php

\*/

namespace Drupal\lotus\Controller;

use Drupal\Core\Controller\ControllerBase;

class LotusController extends ControllerBase {

public function content() {

return array(

'#theme' => 'lotus\_template',

'#test\_var' => $this->t('Test Value'),

);

}

}

In the template folder create the lotus-template.html.twig file:

<p> This is the lotus template with a value of {{ test\_var }} </p>

**Passing configuration values from PHP to a JavaScript library.**

## Drupal 8 method

In the lotus.libraries.yml file, you must declare a dependency to drupalSettings:

lotus-js:

version: 1.x

js:

js/lotus.js: {}

dependencies:

- core/jquery

- core/drupalSettings

In the lotus.module file:

function lotus\_preprocess\_html(&$variables) {

$lotus\_height = '300px';

//Add a JS library

$variables['#attached']['library'][] = 'lotus/lotus-js';

$variables['#attached']['drupalSettings']['lotus']['lotusJS']['lotus\_height'] = $lotus\_height;

}

So in your module\_preprocess\_html you should write

$variables['#attached']['drupalSettings'][‘module\_name’]['variable\_you\_want\_to\_pass']['lotus\_height'] = $lotus\_height;

}

In the lotus.js file:

(function ($, Drupal, drupalSettings) {

Drupal.behaviors.LotusBehavior = {

attach: function (context, settings) {

// can access setting from 'drupalSettings';

var lotusHeight = drupalSettings.lotus.lotusJS.lotus\_height;

$('lotusElement').css('height', lotusHeight);

}

};

})(jQuery, Drupal, drupalSettings);

**Cache for authenticated users in drupal 8?**

<https://www.drupal.org/docs/8/core/modules/dynamic-page-cache/overview>

# **Dynamic Page Cache overview**

Last [updated](https://www.drupal.org/node/2541358/discuss) on

14 June 2018

Drupal 8 provides the Dynamic Page Cache module that is recommended for websites of all sizes. It caches pages minus the personalized parts, and is therefore useful for all users (both anonymous & authenticated).

We can find this core module at: **core/modules/dynamic\_page\_cache**.

This feature improves performance because it makes it possible to cache pages with dynamic content. Pages requested by users (anonymous or authenticated) are stored the first time they are requested and can then reused. Personalized parts are excluded: they are [turned into placeholders automatically](https://www.drupal.org/developing/api/8/render/arrays/cacheability/auto-placeholdering). Depending on your site configuration and the complexity of particular pages, Dynamic Page Cache may significantly increase the speed of your site, even for authenticated users.

**how to share a table between 2 sites in multi site enviroment**

**how to create your own entity?**

We do two things, we **extend** an existing **[ContentEntityBase](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Entity!ContentEntityBase.php/class/ContentEntityBase/8)** class that already has the necessary methods to interact with the DB, and **implement** an **[ContentEntityInterface](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Entity!ContentEntityInterface.php/interface/ContentEntityInterface/8)** to describe...

the methods that we need to access our database. It does NOT describe in any way HOW we achieve that. That's what the IMPLEMENTing class does. We can IMPLEMENT this interface as many times as we need in as many different ways as we need. We can then switch between implementations of the interface without impact to our code because the interface defines how we will use it regardless of how it actually works. - <https://secure.php.net/manual/en/language.oop5.interfaces.php>

class Advertiser extends ContentEntityBase implements ContentEntityInterface {

**how to create rest api in drupal 8?**

**src\Plugin\rest\resource\ExampleGetRestResource.php**

|  |
| --- |
|  |
| **class ExampleGetRestResource extends ResourceBase {** |
|  |

**POST RESt api**

|  |
| --- |
| <?php |
|  |  |
|  | /\*\* |
|  | \* @file |
|  | \* Contains Drupal\custom\_rest\Plugin\rest\resource\custom\_rest. |
|  | \*/ |
|  |  |
|  | namespace Drupal\custom\_rest\Plugin\rest\resource; |
|  |  |
|  | use Drupal\Core\Session\AccountProxyInterface; |
|  | use Drupal\rest\Plugin\ResourceBase; |
|  | use Drupal\rest\ResourceResponse; |
|  | use Symfony\Component\DependencyInjection\ContainerInterface; |
|  | use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException; |
|  | use Symfony\Component\HttpKernel\Exception\HttpException; |
|  | use Psr\Log\LoggerInterface; |
|  |  |
|  | /\*\* |
|  | \* Provides a resource to get view modes by entity and bundle. |
|  | \* |
|  | \* @RestResource( |
|  | \* id = "custom\_rest\_resource", |
|  | \* label = @Translation("Custom rest resource"), |
|  | \* uri\_paths = { |
|  | \* "canonical" = "//api/custom" |
|  | \* } |
|  | \* ) |
|  | \*/ |
|  | class CustomRestResource extends ResourceBase { |
|  | /\*\* |
|  | \* A current user instance. |
|  | \* |
|  | \* @var \Drupal\Core\Session\AccountProxyInterface |
|  | \*/ |
|  | protected $currentUser; |
|  |  |
|  | /\*\* |
|  | \* Constructs a Drupal\rest\Plugin\ResourceBase object. |
|  | \* |
|  | \* @param array $configuration |
|  | \* A configuration array containing information about the plugin instance. |
|  | \* @param string $plugin\_id |
|  | \* The plugin\_id for the plugin instance. |
|  | \* @param mixed $plugin\_definition |
|  | \* The plugin implementation definition. |
|  | \* @param array $serializer\_formats |
|  | \* The available serialization formats. |
|  | \* @param \Psr\Log\LoggerInterface $logger |
|  | \* A logger instance. |
|  | \* @param \Drupal\Core\Session\AccountProxyInterface $current\_user |
|  | \* A current user instance. |
|  | \*/ |
|  | public function \_\_construct( |
|  | array $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | array $serializer\_formats, |
|  | LoggerInterface $logger, |
|  | AccountProxyInterface $current\_user) { |
|  | parent::\_\_construct($configuration, $plugin\_id, $plugin\_definition, $serializer\_formats, $logger); |
|  |  |
|  | $this->currentUser = $current\_user; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* {@inheritdoc} |
|  | \*/ |
|  | public static function create(ContainerInterface $container, array $configuration, $plugin\_id, $plugin\_definition) { |
|  | return new static( |
|  | $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | $container->getParameter('serializer.formats'), |
|  | $container->get('logger.factory')->get('rest'), |
|  | $container->get('current\_user') |
|  | ); |
|  | } |
|  | /\*\* |
|  | \* Responds to POST requests. |
|  | \* |
|  | \* Returns a list of bundles for specified entity. |
|  | \* |
|  | \* @throws \Symfony\Component\HttpKernel\Exception\HttpException |
|  | \* Throws exception expected. |
|  | \*/ |
|  | public function post() { |
|  |  |
|  | // You must to implement the logic of your REST Resource here. |
|  | // Use current user after pass authentication to validate access. |
|  |  |
|  | /\* |
|  | if(!$this->currentUser->hasPermission($permission)) { |
|  | throw new AccessDeniedHttpException(); |
|  | } |
|  | \*/ |
|  |  |
|  | // Throw an exception if it is required. |
|  | // throw new HttpException(t('Throw an exception if it is required.')); |
|  | return new ResourceResponse("Implement REST State POST!"); |
|  | } |
|  |  |
|  | } |

**Get rest api**

|  |
| --- |
| <?php |
|  |  |
|  | namespace Drupal\example\_rest\Plugin\rest\resource; |
|  |  |
|  | use Drupal\Core\Session\AccountProxyInterface; |
|  | use Drupal\rest\Plugin\ResourceBase; |
|  | use Drupal\rest\ResourceResponse; |
|  | use Symfony\Component\DependencyInjection\ContainerInterface; |
|  | use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException; |
|  | use Psr\Log\LoggerInterface; |
|  |  |
|  | /\*\* |
|  | \* Provides a resource to get view modes by entity and bundle. |
|  | \* |
|  | \* @RestResource( |
|  | \* id = "example\_get\_rest\_resource", |
|  | \* label = @Translation("Example get rest resource"), |
|  | \* uri\_paths = { |
|  | \* "canonical" = "/example-rest" |
|  | \* } |
|  | \* ) |
|  | \*/ |
|  | class ExampleGetRestResource extends ResourceBase { |
|  |  |
|  | /\*\* |
|  | \* A current user instance. |
|  | \* |
|  | \* @var \Drupal\Core\Session\AccountProxyInterface |
|  | \*/ |
|  | protected $currentUser; |
|  |  |
|  | /\*\* |
|  | \* Constructs a Drupal\rest\Plugin\ResourceBase object. |
|  | \* |
|  | \* @param array $configuration |
|  | \* A configuration array containing information about the plugin instance. |
|  | \* @param string $plugin\_id |
|  | \* The plugin\_id for the plugin instance. |
|  | \* @param mixed $plugin\_definition |
|  | \* The plugin implementation definition. |
|  | \* @param array $serializer\_formats |
|  | \* The available serialization formats. |
|  | \* @param \Psr\Log\LoggerInterface $logger |
|  | \* A logger instance. |
|  | \* @param \Drupal\Core\Session\AccountProxyInterface $current\_user |
|  | \* A current user instance. |
|  | \*/ |
|  | public function \_\_construct( |
|  | array $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | array $serializer\_formats, |
|  | LoggerInterface $logger, |
|  | AccountProxyInterface $current\_user) { |
|  | parent::\_\_construct($configuration, $plugin\_id, $plugin\_definition, $serializer\_formats, $logger); |
|  |  |
|  | $this->currentUser = $current\_user; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* {@inheritdoc} |
|  | \*/ |
|  | public static function create(ContainerInterface $container, array $configuration, $plugin\_id, $plugin\_definition) { |
|  | return new static( |
|  | $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | $container->getParameter('serializer.formats'), |
|  | $container->get('logger.factory')->get('example\_rest'), |
|  | $container->get('current\_user') |
|  | ); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Responds to GET requests. |
|  | \* |
|  | \* Returns a list of bundles for specified entity. |
|  | \* |
|  | \* @throws \Symfony\Component\HttpKernel\Exception\HttpException |
|  | \* Throws exception expected. |
|  | \*/ |
|  | public function get() { |
|  |  |
|  | // You must to implement the logic of your REST Resource here. |
|  | // Use current user after pass authentication to validate access. |
|  | if (!$this->currentUser->hasPermission('access content')) { |
|  | throw new AccessDeniedHttpException(); |
|  | } |
|  | $entities = \Drupal::entityTypeManager() |
|  | ->getStorage('node') |
|  | ->loadMultiple(); |
|  | foreach ($entities as $entity) { |
|  | $result[$entity->id()] = $entity->title->value; |
|  | } |
|  |  |
|  | $response = new ResourceResponse($result); |
|  | $response->addCacheableDependency($result); |
|  | return $response; |
|  | } |
|  |  |
|  | } |

**what is drupal queue / drupal batch**

<http://karimboudjema.com/en/drupal/20180807/create-queue-controller-drupal8>

Queues are particularly important when we need to stash some tasks for later processing. To do so, we are going to put some tasks or data in a queue (create the queue) and later we will process those tasks with a QueueWorker plugin (process the queue), usually triggered by cron.

There are several ways to create a queue:  
- With a form class  
- With a controller class  
- With a hook\_cron() function  
   
To process the queue, we also have different options:  
- As a cron process with a QueueWorker plugin  
- As a batch process also with QueueWorker plugin but extending a base plugin  
- As a batch process claiming each item of the queue in a service or in a controller

Acquia DAM uses drupal queue.

Drupal queue maintain a table.

**What is final keyword?**

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with *final*. If the class itself is being defined final then it cannot be extended.

Example #1 Final methods example

<?php  
class BaseClass {  
   public function test() {  
       echo "BaseClass::test() called\n";  
   }  
     
   final public function moreTesting() {  
       echo "BaseClass::moreTesting() called\n";  
   }  
}  
  
class ChildClass extends BaseClass {  
   public function moreTesting() {  
       echo "ChildClass::moreTesting() called\n";  
   }  
}  
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()  
?>

Example #2 Final class example

<?php  
final class BaseClass {  
   public function test() {  
       echo "BaseClass::test() called\n";  
   }  
  
   // Here it doesn't matter if you specify the function as final or not  
   final public function moreTesting() {  
       echo "BaseClass::moreTesting() called\n";  
   }  
}  
  
class ChildClass extends BaseClass {  
}  
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)  
?>

**what is private and protected**

**what is abstract and interface?**

An **interface** is a contract. An **interface** defines what a class can do, without saying anything about how the class will do it. All the methods in an **interface** is public and **abstract**. An **abstract** class is used to define blueprint for a child classes. The **abstract** class is only created for an inheritance.

**how to do route alter in drupal 8?**

<https://www.drupal.org/docs/8/api/routing-system/altering-existing-routes-and-adding-new-routes-based-on-dynamic-ones>

You can alter existing routes by implementing the [alterRoutes(RouteCollection $collection)](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Routing%21RouteSubscriberBase.php/function/RouteSubscriberBase%3A%3AalterRoutes/8) method of this class.

**class RouteSubscriber extends RouteSubscriberBase {**

class RouteSubscriber extends RouteSubscriberBase {

/\*\*

\* {@inheritdoc}

\*/

protected function alterRoutes(RouteCollection $collection) {

// Define custom access for '/user/login'.

if ($route = $collection->get('user.login')) {

$route->setRequirement('\_custom\_access', 'Drupal\example\Access\StandardAccessCheck::access');

}

// Define custom access for '/user/logout'.

if ($route = $collection->get('user.logout')) {

$route->setRequirement('\_custom\_access', 'example.services\_access\_checker::access');

}

}

**how to pass variables from drupal to JS?**

<https://docs.acquia.com/tutorials/fast-track-drupal-8-coding/add-custom-variable-drupalsettings/>

## Drupal 8 method

In the lotus.libraries.yml file, you must declare a dependency to drupalSettings:

lotus-js:

version: 1.x

js:

js/lotus.js: {}

dependencies:

- core/jquery

- core/drupalSettings

In the lotus.module file:

function lotus\_preprocess\_html(&$variables) {

$lotus\_height = '300px';

//Add a JS library

$variables['#attached']['library'][] = 'lotus/lotus-js';

$variables['#attached']['drupalSettings']['lotus']['lotusJS']['lotus\_height'] = $lotus\_height;

}

In the lotus.js file:

(function ($, Drupal, drupalSettings) {

Drupal.behaviors.LotusBehavior = {

attach: function (context, settings) {

// can access setting from 'drupalSettings';

var lotusHeight = drupalSettings.lotus.lotusJS.lotus\_height;

$('lotusElement').css('height', lotusHeight);

}

};

})(jQuery, Drupal, drupalSettings);

**based on the value of the field in node we need to call page template?**

**how to write service in drupal 8?**

In **Drupal 8** speak, a **service** is any object managed by the **services** container. **Drupal 8** introduces the concept of **services** to decouple reusable functionality and makes these **services** pluggable and replaceable by registering them with a **service** container.

 As of **Drupal** 8, **dependency injection** is the preferred method for accessing and using services and should be used whenever possible.

**git fetch and pull difference?**

**how you can bring changes from 1 branch to another?**

**what are the ports of http and https?**

**do you know what is PO files?  
  
what is t function?  
  
how to add div inside a div?  
  
how do you add field in existing table by using hooks?  
  
what is scope in drupal add js?  
  
how to export configuraton in git?  
  
which file is called first when calling a page in drupal?  
  
what is hook update N?**  
You implement hook\_update\_N(), and inside that function, you can add db\_add\_field(). You should also add the field definition to your implementation of hook\_schema().  
  
  
  
difference between drupal 7 and drupal 8?  
  
what is migrate path UI drupal 8?  
  
what is the order of template content type, node type and field type?  
  
how you will transfer a feature which is not transferred by migrate UI path?  
  
how to create a form in drupal 8?

what are mandatory fiels in info.yml file?  
name and type

what s difference between git fetch and git pull?  
  
module\_invoke\_all Invokes a hook in all enabled modules that implement it.  
  
module\_invoke Invokes a hook in a particular module.  
  
Invokes a hook in a particular module.  
  
Invokes a hook in a particular module.  
  
WHAT IS THE T() FUNCTION?