



# **DRUPAL DEVELOPER DAYS**

**ATHENS 2026**



# Refactor to #[Hook]

**Object oriented hooks  
less magic, more dependency injection**

---

---



[drupal.community/@bircher](https://drupal.community/@bircher)



## Fabian Bircher

Senior Software Engineer





# What is a hook?

Hooks are functions or methods that a module defines, which are discovered and called at specific times to alter or add to the base behavior or data.

<https://api.drupal.org/api/drupal/core%21core.api.php/group/hooks/11.x>





# Q&A with Dries @ DDD Leuven 2025

Q: Is there a thing in Drupal core or contrib right now that doesn't get the attention it should be getting?

A: One thing that went in that I thought was interesting and I haven't really heard people talk about it, is this thing called "hooks as classes".

In Atlanta nobody mentioned it. So check that out. I think that was a pretty interesting commit.

<https://youtu.be/LK7QkKZJobA?feature=shared&t=2600>





# #[Attribute]

PHP attributes provide structured, machine-readable metadata for classes, methods, functions, parameters, properties, and constants.

Attributes provide a declarative way to annotate code with metadata.

<https://www.php.net/manual/en/language.attributes.overview.php>





# Example

```
#[ \SensitiveParameter ]
```

```
#[ \AllowDynamicProperties ]
```

```
#[ \Override ]
```

```
#[ \ReturnTypeWillChange ]
```

```
#[ \Deprecated(message: "use ... instead", since: "1.5") ]
```



# Example

```
namespace Drupal\Core>Password;

interface PasswordInterface {
    public function hash([\SensitiveParameter] $password);
    public function check([\SensitiveParameter] $password,
        #[\SensitiveParameter] $hash);
    public function needsRehash([\SensitiveParameter] $hash);
}
```



**hook\_ -> /\*\* Plugin \*\*/ -> #[Plugin]**

Many hook use-cases were converted to plugins in Drupal 8

Drupal 8 plugins used annotations (metadata in a comment)

Since php8 (Drupal 10.2-11.2) annotations are replaced with attributes

Transition required in Drupal 13

<https://www.drupal.org/node/3395575>





# Plugin with attribute example

```
namespace Drupal\help\Plugin\Block;  
  
use Drupal\Core\Block\Attribute\Block;  
use Drupal\Core\Block\BlockBase;  
  
#[Block(  
  id: "help_block",  
  admin_label: new TranslatableMarkup("Help"),  
  forms: ['settings_tray' => FALSE]  
)]  
class HelpBlock extends BlockBase {}
```



# Problems with procedural hooks

- Magic name can result in name conflicts

```
module_name_hook_name
```

- No dependency injection makes tests harder to write
- all functions in .module file are always loaded



# Event Subscriber everything?

Effort to use event subscribers more, BUT: More boilerplate 🤖

Especially dynamic hooks don't work well as event subscribers.

You can't implement optional Event Subscribers because the class won't exist, unlike a D7 hook which just won't execute.

Event subscribers behave differently (ex: can be stopped) => no 1 to 1



# What do we want?

Class with dependency injection for easier testing

No ambiguity about which hook is implemented

As little boilerplate as possible. (Autowiring)

As straight forward conversion from previous system as possible

Optional hooks

Re-ordering hooks



# Change Records

[#3442349](#): Support for object oriented hook implementations using autowired services

[#3551652](#): Hooks in themes can now be OOP





# Implement a hook

Directly on a method in a class

```
#[Hook('user_cancel')]  
public function userCancel(...){}
```

On the class, specifying the function

```
#[Hook('user_cancel', 'userCancel')]  
class BeachVibes {  
    public function userCancel(...) {}  
}
```

On the class, in an invoke method

```
#[Hook('user_cancel')]  
class BeachVibes {  
    public function __invoke(...) { // This is the hook. }  
}
```



# Multiple hooks, one function

```
function _update_external_user_management_system($role) {  
    // Update external user management system.  
}  
  
function custom_user_role_insert(Role $role) {  
    _update_external_user_management_system($role);  
}  
  
function custom_comment_update(Role $role) {  
    _update_external_user_management_system($role);  
}  
  
function custom_comment_manipulation(Role $role) {  
    _update_external_user_management_system($role);  
}
```



# Multiple hooks, one function

```
class CustomManageRoles {  
    #[Hook('user_role_insert')]  
    #[Hook('user_role_update')]  
    #[Hook('user_role_delete')]  
    public function updateExternalUserManagementSystem() {  
        // Update external user management system.  
    }  
}
```



# Ordering

## Simple

```
#[Hook('user_cancel', order: Order::First)]  
#[Hook('user_cancel', order: Order::Last)]
```

## Specific

```
#[Hook('form_filter_format_form_alter',  
  order: new OrderAfter(  
    modules: ['editor', 'media'],  
  )  
)]  
public function formFilterFormatFormAlter(...) {
```



# Ordering

## Reordering

```
#[ReorderHook('entity_presave',  
  class: ContentModerationHooks::class,  
  method: 'entityPresave',  
  order: new OrderBefore(['field_encrypt'])  
)]  
public function entityPresave(EntityInterface $entity): void {  
    // Steal all your private data.  
}
```



# Removal

```
// Remove a help for Layout Builder without removing other useful help items.
#[RemoveHook('help',
  class: LayoutBuilderHooks::class,
  method: 'help'
)]

// I enjoy chaos.
#[RemoveHook('query_node_access_alter',
  class: NodeDatabaseHooks::class,
  method: 'queryNodeAccessAlter'
)]
```



# Backwards compatibility

For contrib/custom modules to simultaneously support attribute and legacy hooks (10.1 to 11.0)

```
# [LegacyHook]
```

```
use Drupal\Core\Hook\Attribute\LegacyHook;
use Drupal\node\Hook\NodeHooks;

#[LegacyHook]
function node_user_cancel($edit, UserInterface $account, $method) {
    return \Drupal::service(NodeHooks::class)->userCancel($edit, $account, $method);
}
```

```
services:
  Drupal\node\Hook\NodeHooks:
    class: Drupal\node\Hook\NodeHooks
    autowire: true
```

```
namespace Drupal\node\Hook;

use Drupal\Core\Hook\Attribute\Hook;
use Drupal\user\UserInterface;

class NodeHooks {
    #[Hook('user_cancel')]
    public function userCancel($edit, UserInterface $account, $method) {
        // User cancellation processing.
    }
}
```



# Performance improvement

When all hooks are converted:

```
module_name.services.yml
```

```
parameters:
```

```
  module_name.skip_procedural_hook_scan: true
```



# Remaining procedural

`hook_install()`

`hook_uninstall()`

`hook_schema()`

`hook_install_tasks()`

`hook_install_tasks_alter()`

`hook_post_update_NAME()`

`hook_removed_post_updates()`

`hook_update_dependencies()`

`hook_update_last_removed()`



# Legacy hooks

`hook_hook_info()`

`hook_module_implements_alter()`

`hook_requirements()`



# hook\_requirements()

```
# [Hook('runtime_requirements')]
```

<https://www.drupal.org/node/3490851>

```
# [Hook('update_requirements')]
```

<https://www.drupal.org/node/3490852>

```
Drupal\Core\Extension\InstallRequirementsInterface
```

<https://www.drupal.org/node/3492429>



# template\_preprocess\_HOOK

Rather than magically named functions such as `template_preprocess_container` these functions are now set as callbacks on the `hook_theme()` defining the HOOK.

<https://www.drupal.org/node/3504125>





# Refactor using Rector

```
composer require --dev palantirnet/drupal-rector
```

```
cp vendor/palantirnet/drupal-rector/rector.php .
```

```
$rectorConfig->rule(\DrupalRector\Rector\Convert\HookConvertRector::class);
```

```
vendor/bin/rector process path/to/my_module
```

<https://www.noreiko.com/index.php/blog/converting-hooks-oo-methods-made-easy>





# Demo





# Thank you!

Feel free to reach out:

 <https://nuvole.org>

 bircher



# Thank you for joining us!

