DRUPAL DEVELOPER DAYS

ATHENS 2026

Mago – format, lint and analyze your PHP code

Klaus Purer, Jobiqo

Klaus Purer, aka klausi

About me

- Principal Engineer @ Jobiqo
- Doing Drupal since 2008
- Maintainer: Coder, GraphQL, D7Security
- Skills and interests
  - Information Security
  - Performance
  - Backend development
  - REST, GraphQL, API design
  - PHP, Rust

Why Coding Standards?

Formatted code is easier to review and to spot errors

Bad Example (Bypass Vulnerability)

```php
<?php
$isAdmin = FALSE;
$hasAccess = FALSE;

// Intended: only admins get access.
if ($isAdmin)
 error_log('Granting admin access');
 $hasAccess = TRUE;

var_dump($hasAccess); // bool(true)
```

Good, with braces

```php
<?php
$isAdmin = FALSE;
$hasAccess = FALSE;

// Intended: only admins get access.
if ($isAdmin) {
 error_log('Granting admin access');
 $hasAccess = TRUE;
}
var_dump($hasAccess); // bool(false)
```

Why automating Coding Standards?

- Humans do not see all coding standard violations during code review
- Developers take feedback from humans personally
- Developers happily fix any error a tool outputs
- Many coding standards can be fixed automatically by tools such as PHPCS and Mago

PHP Codesniffer (PHPCS) and Coder

- PHP CodeSniffer is a PHP CLI tool, install with Composer
- provides sniffs (rules) to check coding standards
- Drupal's Coder is a set of sniffs that check for Drupal's coding standards
- Used by Drupal core and is executed on drupal.org Gitlab runners
- Setting up a phpcs.xml.dist config file helps with execution
- Can be integrated with your editor, for example PHPStorm or VSCode

Install: `composer require --dev drupal/coder`

Run it locally: `./vendor/bin/phpcs`

Example phpcs.xml.dist

```
<?xml version="1.0" encoding="UTF-8"?>
<ruleset name="jobiqo">
 <description>PHP CodeSniffer configuration for jobiqo development.</description>
 <file>web/profiles/jobiqo</file>
 <file>web/modules</file>
 <!--- Exclude contrib modules. -->
 <exclude-pattern>modules/contrib</exclude-pattern>
 <file>../behat</file>
 <arg name="extensions" value="inc,install,module,php,profile,test,theme,yml"/>

 <rule ref="Drupal">
   <!-- Disable Drupal deprecation tag format in favor of Jobiqo format. -->
   <exclude name="Drupal.Commenting.Deprecated.IncorrectTextLayout"/>
   <exclude name="Drupal.Commenting.Deprecated.DeprecatedMissingSeeTag"/>
 </rule>
 <rule ref="DrupalPractice"/>
  <!-- Jobiqo rules -->
 <rule ref="phpcs/Jobiqo"/>
 <!-- Use statement sorting was removed from Coder, but we want to keep it.
 See https://www.drupal.org/project/coder/releases/8.3.26 -->
 <rule ref="SlevomatCodingStandard.Namespaces.AlphabeticallySortedUses">
   <properties>
     <property name="caseSensitive" value="false"/>
   </properties>
 </rule>
</ruleset>
```

Example phpcs output

```
❯ ddev phpcs

......E..... 12 / 12 (100%)



FILE: /var/www/html/backend/web/profiles/jobiqo/modules/jobiqo_common/jobiqo_common.module
----------------------------------------------------------------------------------------------------------------------------------
---
FOUND 3 ERRORS AFFECTING 3 LINES
----------------------------------------------------------------------------------------------------------------------------------
---
 35 | ERROR | [x] Line indented incorrectly; expected 2 spaces, found 3 (Drupal.WhiteSpace.ScopeIndent.IncorrectExact)
 36 | ERROR | [x] Line indented incorrectly; expected 2 spaces, found 3 (Drupal.WhiteSpace.ScopeIndent.IncorrectExact)
 43 | ERROR | [x] Opening brace should be on the same line as the declaration
(Drupal.Functions.MultiLineFunctionDeclaration.BraceOnNewLine)
----------------------------------------------------------------------------------------------------------------------------------
---
PHPCBF CAN FIX THE 3 MARKED SNIFF VIOLATIONS AUTOMATICALLY
----------------------------------------------------------------------------------------------------------------------------------
---

Time: 10.44 secs; Memory: 40.41MB
```

Run PHPCS faster in parallel mode

Use the number of your CPU cores with nproc:

```
./vendor/bin/phpcs  --parallel="$(nproc)"
```

Use a ddev shortcut command to run “ddev phpcs” in parallel mode.

Put this into .ddev/commands/web/phpcs.sh:

```
#!/usr/bin/env bash
set -e
cd /var/www/html/backend
./vendor/bin/phpcs -s -p --parallel="$(nproc)" "$@"
```

Run PHPCS in CI

- On drupal.org Gitlab CI: runs per default in standard Gitlab template
- If you use DDEV: add “ddev phpcs” from the previous slide as testing step in your CI config
- For advanced reporting invoke phpcs with options

```
--report-junit=junit.xml --report-full --report-summary”
```

PHP Code Beautifier (PHPCBF)

CLI tool to automatically fix all fixable errors, yay!

```
./vendor/bin/phpcbf  --parallel="$(nproc)"
```

Use a ddev shortcut command to run “ddev phpcbf” in parallel mode.

Put this into .ddev/commands/web/phpcbf.sh:

```
#!/usr/bin/env bash
set -e
cd /var/www/html/backend
./vendor/bin/phpcbf -s -p --parallel="$(nproc)" "$@"
```

Ignoring some errors in code

Ignoring in phpcs.xml.dist:

```
<rule ref="DrupalPractice.Objects.GlobalClass">
  <exclude-pattern>src/Plugin/GraphQL/Entity/Fields/Image/ImageDerivative.php</exclude-pattern>
</rule>
```

Ignoring directly in code:

```
class Search extends FilterPluginBase {
 // phpcs:ignore Drupal.NamingConventions.ValidVariableName.LowerCamelName
 public string $search_score;
}
```

Getting started with PHPCS checking

On an existing code base you will get a lot of errors.

1. First run phpcbf manually to get the easy autofixes out of the way
    - This will generate a large change that is very long to review, but only fixes small formatting issues
2. Run phpcs and temporarily disable all violating sniffs in phpcs.xml.dist until phpcs passes
    - LLMs are good at generating the config sniff excludes for you, but carefully review
    - Enable PHPCS checking in CI now to avoid any regressions
3. Enable sniffs one by one, fixing violations (or using phpcs:ignore in code)
    - LLMs are good at generating scripts to do bulk fixes, but carefully review
4. Repeat 3. until all sniffs are enabled and enforced

This process is used in Drupal core as well to fix Coding Standards gradually.

Linting with php -l

- Check your PHP code for syntax errors
- Find the most obvious mistakes before you deploy your code to production
- Note: this will not find runtime errors such as an undefined function being called!

Linting with php -l

ddev exec ../scripts/build/php-lint.sh

```
#!/bin/bash
# Step into the root project folder
CWD=$(dirname $0)
cd ${CWD}/../..
EXIT=0
for FILE in $(find ../behat scripts web/profiles/jobiqo -regextype posix-egrep -regex
".*\.(php|module|install|inc|theme|test|profile)$")
do
 if (php -l $FILE | grep -v "No syntax errors detected"); then
   EXIT=1
 fi
done
exit $EXIT
```

Why static analysis?

- Beyond syntax we can check the semantics and structure of code
- Check that you only call functions, methods or classes that actually exist
- Check that PHP's types are correctly used
- Help static analysis: use type hints wherever you can:
  - function parameters
  - function return types
  - Doxygen type comments: write "@param int[] $ids" instead of just array
  - Use ::class syntax instead of class names in strings
- Static analysis is guardrail for LLM generated code as well

Bad, no types

```
/**
* Example function.
*
* @param array $ids
*   An array of IDs.
* @param string $searchTerm
*   A search term.
* @param RouteCollection $collection
*   A collection of routes.
*/
function example($ids, $searchTerm, $collection) {
  if ($route = $collection->get('entity_print.view')) {
    $route->setDefault('_controller', '\Drupal\mymodule\ExampleController::viewPrint');
    return TRUE;
  }
  return FALSE;
}
```

Good, with types

```php
/**
* Example function.
*
* @param int[] $ids
*   An array of IDs.
* @param string $searchTerm
*   A search term.
* @param RouteCollection $collection
*   A collection of routes.
*/
function example(array $ids, string $searchTerm, RouteCollection $collection): bool {
  if ($route = $collection->get('entity_print.view')) {
    $route->setDefault('_controller', ExampleController::class . '::viewPrint');
    return TRUE;
  }
  return FALSE;
}
```

PHPStan

- PHPStan is a PHP CLI tool, install with Composer https://phpstan.org
- provides rules to do static analysis
- Used by Drupal core and is executed on drupal.org Gitlab runners
- Use levels from 0 to 9 for the amount of detail that will be checked
- We found that level 6 is a good level for Drupal sites
- Setting up a phpstan.neon config file is necessary

Install: `composer require --dev phpstan/phpstan`

Run it locally: `./vendor/bin/phpstan analyse`

Install phpstan-drupal and other useful helpers in composer.json

```
"require-dev": {
    "jangregor/phpstan-prophecy": "^2",
    "mglaman/phpstan-drupal": "^2",
    "phpstan/extension-installer": "^1.4.3",
    "phpstan/phpstan": "^2",
    "phpstan/phpstan-deprecation-rules": "^2",
    "phpstan/phpstan-phpunit": "^2"
},
```

Example phpstan.neon part 1

```
parameters:
 level: 6
 # We want to keep our code and PHPStan config file clean of unnecessary ignore
 # instructions, this setting ensures that.
 reportUnmatchedIgnoredErrors: true
 paths:
   - web/profiles/jobiqo
   - web/modules
   - ../behat
   - phpstan/src
 # Exclude contrib modules.
 excludePaths:
   - web/modules/contrib/*
 customRulesetUsed: true
 fileExtensions:
   - inc
   - install
   - module
   - php
 # Not all Drupal core and contrib phpdoc types are fully correct, so PHPStan
 # would give wrong advice.
 treatPhpDocTypesAsCertain: false
```

Example phpstan.neon part 2

```
ignoreErrors:
   # Many Drupal classes are iterable where we cannot specify a value type.
   - identifier: missingType.iterableValue
   # PHPCS doesn't seem to be able to handle generics, so have to ignore those.
   - identifier: missingType.generics
   # @todo Ignore phpstan-drupal extension's rules for now, activate later.
   - '#\Drupal calls should be avoided in classes, use dependency injection instead#'
   # new static() is a best practice in Drupal, so we cannot fix that.
   - "#^Unsafe usage of new static\\(\\)\\.$#"
   # Drupal core documentation bugs that should be fixed in Drupal core.
   - "#^Parameter \\#1 \\$values of method Drupal\\\\Core\\\\Field\\\\FieldItemBase\\:\\:setValue\\(\\)
expects array\\|null#"
 universalObjectCratesClasses:
   # Drupal allows object property access to custom fields, so we need
   # exceptions for entity and field classes.
   - Drupal\Core\Entity\EntityInterface
   - Drupal\Core\Field\FieldItemInterface
   - Drupal\Core\Field\FieldItemListInterface
   - Drupal\Core\TypedData\TypedDataInterface
   - Drupal\views\ResultRow
```

Example phpstan.neon part 3

```
drupal:
  entityMapping:
    block:
      class: Drupal\block\Entity\Block
      storage: Drupal\Core\Config\Entity\ConfigEntityStorage
    node:
      class: Drupal\node\Entity\Node
      storage: Drupal\node\NodeStorage
    search_api_index:
      class: Drupal\search_api\Entity\Index
      storage: Drupal\search_api\Entity\SearchApiConfigEntityStorage
    search_api_server:
      class: Drupal\search_api\Entity\Server
      storage: Drupal\search_api\Entity\SearchApiConfigEntityStorage
    taxonomy_term:
      class: Drupal\taxonomy\Entity\Term
      storage: Drupal\taxonomy\TermStorage
    user:
      class: Drupal\user\Entity\User
      storage: Drupal\user\UserStorage
```

PHPStan example output

```
❯ ./vendor/bin/phpstan
Note: Using configuration file /home/klausi/workspace/jd/backend/phpstan.neon.
 2535/2535 [▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓] 100%

 ------ ---------------------------------------------------------------------------------------------
  Line   backend/web/profiles/jobiqo/modules/jobiqo_translation/src/Locale/PoDatabaseWriter.php
 ------ ---------------------------------------------------------------------------------------------
  33     Method Drupal\jobiqo_translation\Locale\PoDatabaseWriter::writeItems() has parameter $count with no type specified.
         🪪  missingType.parameter
  50     Call to an undefined method Drupal\Component\Gettext\PoItem::getContex().
         🪪  method.notFound
 ------ ---------------------------------------------------------------------------------------------

 [ERROR] Found 2 errors
```

Ignoring some errors in code

Ignoring directly in code:

```
/**
 * {@inheritdoc}
 *
 * @phpstan-ignore missingType.parameter
 */
public function writeItems(PoReaderInterface $reader, $count = -1): void {
```

Doc type annotations instead of ignores

- PHPStan reads @var comments that override PHP types
- Drupal's entities and fields are dynamic: PHstan cannot get more specific type information
- Developers decide with comments that a variable must be of type X

```
foreach ($order->getItems() as $orderItem) {
  /** @var \Drupal\commerce_product\Entity\ProductVariationInterface $productVariation */
  $productVariation = $orderItem->getPurchasedEntity();
  /** @var \Drupal\commerce_product\Entity\ProductInterface $product */
  $product = $productVariation->product_id->entity;
  if ($product->hasField('field_commerce_features')) {
    …
  }
}
```

Getting started with PHPStan checking

On an existing code base you will get a lot of errors.

1. Enable level 0, fix errors
    - LLMs are good at generating scripts to do bulk fixes, but carefully review
    - Enalbe PHPStan checking in CI now to avoid any regressions
2. Enable level 1, generate PHPStan baseline file
    - For any current violation there will be an ignore statement in phpstan-baseline.php
    - Any new or changed code will be validated, protecting you again regressions on level 1
3. Remove baseline, fix level 1
    - LLMs are good at generating scripts to do bulk fixes, but carefully review
4. Repeat from 2. by increasing level number

Level 6 is a good goal to reach, higher levels are stricter and can be inconvenient

Mago

- Mago is a Rust CLI tool
  - Code Formatter (comparable to PHPCS)
  - Linter (comparable to PHP Mess Detector and some PHPCS sniffs)
  - Static Analyzer (comparable to PHPStan)
- The advantage is performance: runs in seconds, not minutes
- Drupal support is very basic and not mature yet

Install: `composer require --dev carthage-software/mago`

Run it locally: `./vendor/bin/mago fmt`

https://mago.carthage.software/

Mago performance on Drupal core

- phpcs (parallel mode): 27 seconds
- mago fmt: 1.3 seconds initial run, 1.1 seconds repeated run (20 times faster)
- phpstan: 1 minute 50 seconds
- mago lint: 1.3 seconds
- mago analyze: 5 seconds (22 times faster)

1 second formatting time means we can always trigger it:

- On every save in your editor
- On every git pre-commit hook
- In continous integration to always push formatter fixes automatically

Example mago.toml config for Drupal core

```toml
version = "1"
php-version = "8.4.0"

[source]
workspace = "."
paths = [
 ".",
 "../composer",
]
includes = ["../vendor"]
excludes = [
 "**/bower_components/**",
 "**/node_modules/**",
]
extensions = ["php", "inc", "module", "profile", "install", "theme"]

[source.glob]
literal-separator = true

[formatter]
preset = "drupal"

[linter]
integrations = ["symfony", "phpunit"]
```

Mago fmt philosophy

```
❯ ../vendor/bin/mago fmt
 INFO Formatted 7178 file(s) successfully.
```

- No output of errors, just rewriting source code
- Formatting is not a human task, it will always be done automatically by mago
- mago fmt is like phpcbf (not phpcs)

From the Mago formatter docs:

“Its primary goal is to end debates over code style. By automating the formatting process, it allows you and your team to stop worrying about whitespace and focus on what truly matters: building great software.”

Mago fmt is incomplete

There is one important feature missing in mago fmt: doc comment formatting

These white space errors are currently not fixed by mago (issue #906):

```
/**
  * Determines if a plugin instance exists.
  *
  *   @param string $instance_id
  *     The ID of the plugin instance to check.
  *
  *@return bool
  *   TRUE if the plugin instance exists, FALSE otherwise.
  */
public function has($instance_id) {
```

Mago lint and analyze

- Mago lint and analyze have a lot of autofix implementations, yay!
- Current state: many issues with dynamic Drupal entity and field types
- No configuration practices or presets for Drupal yet
- Recommendation: disable lint rules in configuration and only apply the ones that work for Drupal

Example mago lint output

```
❯ ../vendor/bin/mago lint lib/Drupal/Core/Entity/BundlePermissionHandlerTrait.php
warning[strict-types]: Missing `declare(strict_types=1);` statement at the beginning of the file.
  ┌─ lib/Drupal/Core/Entity/BundlePermissionHandlerTrait.php:1:1
  │
1 │ <?php
  │ ^^^^^
  │
  = The `strict_types` directive enforces strict type checking, which can prevent subtle bugs.
  = Help: Add `declare(strict_types=1);` at the top of your file.

help[prefer-static-closure]: This closure does not use `$this` and should be declared static.
   ┌─ lib/Drupal/Core/Entity/BundlePermissionHandlerTrait.php:30:9
   │
30 │         function (array $perm) use ($bundle) {
   │         ^^^^^^^^ add `static` before this closure keyword
   │
   = Static closures are more memory-efficient and make it clear that `$this` is not used.
   = Help: Add the `static` keyword before `function` to make this closure static.

warning: found 2 issues: 1 warning(s), 1 help message(s)
 = 2 issues contain auto-fix suggestions
```

Example mago analyze output

```
❯ ../vendor/bin/mago analyze

error[malformed-docblock-comment]: Failed to parse function-like docblock comment.
    ┌─ /home/klausi/workspace/drupal-11/composer/Plugin/Scaffold/Operations/ScaffoldFileCollection.php:162:6
    │
162 │     * @deprecated. Called when upgrading from the Core Composer Scaffold plugin
    │       ^^^^^^^^^^^^ Invalid tag name
    │
    = Docblock tag names must contain only letters, numbers, underscores, hyphens, colons, or backslashes.
    = Help: Correct the tag name to use only valid characters (e.g., `@my-custom-tag`).

error[malformed-docblock-comment]: Failed to parse function-like docblock comment.
    ┌─ tests/Drupal/Tests/PerformanceData.php:405:6
    │
405 │     * @@return string[]
    │       ^^^^^^^^ Invalid tag name
    │
    = Docblock tag names must contain only letters, numbers, underscores, hyphens, colons, or backslashes.
    = Help: Correct the tag name to use only valid characters (e.g., `@my-custom-tag`).

error: found 168424 issues: 126644 error(s), 40734 warning(s), 1046 help message(s)
 = 1903 issues contain auto-fix suggestions
```

Ignoring mago errors in code

Ignoring directly in code with @mago-expect
https://mago.carthage.software/fundamentals/suppressing-issues

```
// @mago-expect lint:no-shorthand-ternary,unused-variable
$result = $value ?: 'default';
```

Or in config:

```
[linter.rules]
# Disable a rule completely
ambiguous-function-call = { enabled = false }
```

A baseline to ignore existing issues

```
# Generate a baseline for the linter
mago lint --generate-baseline --baseline mago-lint-baseline.toml

# Generate a baseline for the analyzer
mago analyze --generate-baseline --baseline mago-analysis-baseline.toml

# Run the linter using its baseline
mago lint --baseline mago-lint-baseline.toml

# Run the analyzer using its baseline
mago analyze --baseline mago-analysis-baseline.toml
```

The plan to integrate Mago for Drupal

- Use mago fmt and PHPCS with Coder for everything that mago does not check yet
- Define a new PHPCS Standard for that in Coder: DrupalMago
  - It includes only sniffs that mago does not cover
- Add a new helper CLI command to generate a Mago config from a PHPCS config
- Add a new wrapper CLI command that runs 4 parts:
  - mago fmt
  - mago lint
  - mago analyze
  - phpcs
- Add documentation about Drupal best practices what can be used from Mago

Current state of Mago and Drupal

- Does mago replace PHPCS or PHPStan?
  - No, not yet.
  - Many checks are not available in Mago
  - Many checks in Mago cannot be applied 1:1 to Drupal
- What is the point then?
  - Explore Mago to eventually replace PHPCS and/or PHPStan (will take years)

How to get involved

- Test Mago yourself on any Drupal code, report Mago bugs to https://github.com/carthage-software/mago/issues
- Follow the plan for Coder to create a new DrupalMago standard https://www.drupal.org/project/coder/issues/3585617
- Join the #coding-standards channel on Drupal Slack
- Discuss Coding Standard changes at https://www.drupal.org/project/coding_standards

Thank you!

Feel free to reach out:

https://klau.si

klausi

@klausi@mastodon.social

/in/klausi

Thank you for joining us!