Console Project

Symfony Console Integration for the Derafu Kernel

Anonymous · 09/09/2026 · #php

Symfony Console Integration for the Derafu Kernel

Adds Symfony Console command auto-discovery and execution to any `Derafu\Kernel\MicroKernel`-based Kernel — no per-command registration, no opinion on whether a project reuses its main Kernel or a dedicated console-only one.

What This Package Provides

- `Derafu\Console\DependencyInjection\CommandCompilerPass`: collects every service tagged `console.command` into a `console.command_ids` container parameter.
- `Derafu\Console\ConsoleKernelTrait`: adds command auto-discovery and a `run(): int` method to any class extending `Derafu\Kernel\MicroKernel` (or providing an equivalent `getContainer(): ContainerInterface`).
- `Derafu\Console\Kernel`: a ready-to-use console Kernel for projects that do not need to compose console support into an existing Kernel.
- `Derafu\Console\Runtime`: resolves the real process environment (`APP_ENV`/`APP_DEBUG`) before any Kernel exists — see below.

This package does not depend on `derafu/http` or any HTTP-related package. It has no opinion on whether a project reuses its main application Kernel, a dedicated console-only subclass of it, or a fresh one — that decision belongs entirely to each project.

Installation

```
composer require derafu/console
```

`Runtime`: Resolving `APP_ENV`/`APP_DEBUG` Before the Kernel Exists

A Kernel's own `Derafu\Kernel\Environment::getEnv()` cannot help decide *its own* `$environment`/`$debug` constructor arguments — it's an instance method, and the instance doesn't exist yet. Something has to resolve those two values from the real process environment first.

Reading `$_ENV['APP_ENV']` directly is not reliable: `$_ENV` is only populated when the `variables_order` php.ini directive includes `E`, which is not every PHP installation's default — a stock Homebrew PHP on macOS, for example, ships with `GPCS`, without `E`. `$_SERVER` does carry real process environment variables in the CLI SAPI regardless of that setting, so `Runtime::getApplicationContext()` merges both:

```
use Derafu\Console\Runtime;

$context = Runtime::getApplicationContext();
// ['APP_ENV' => 'prod', 'APP_DEBUG' => false, ...] — plus whatever else
// was already in $_SERVER/$_ENV.
```

Defaults to `prod`/`false` when neither is set — deliberately the *opposite* of `derafu/http`'s `Runtime`, whose `dev`/`true` fallback only matters when an HTTP project has no `.env` of its own (`Derafu\Kernel\Environment::loadEnvironmentVariables()` loads one). A console tool installed via Composer and run as `bin/console` has no such file: the values this class resolves *are* the actual default a fresh install runs with, so it has to be the safe one — a stack trace should never be exposed unless `APP_DEBUG` is explicitly set.

`Runtime::run()` resolves the context, builds the Kernel through a factory, and runs it — this is the actual recommended `bin/console` entry point, not `getApplicationContext()` used by hand:

```
// bin/console
use Derafu\Console\Kernel;
use Derafu\Console\Runtime;

require dirname(__DIR__) . '/vendor/autoload.php';

exit(Runtime::run(fn (array $context): Kernel => new Kernel(
    $context['APP_ENV'],
    (bool) $context['APP_DEBUG'],
)));
```

Usage

A Project With No Other Kernel (Simplest Case)

`Kernel` reads `services.yaml` from the environment's configuration directory. Any service in it whose

class extends `Symfony\Component\Console\Command\Command` is discovered automatically.

```
// bin/console
use Derafu\Console\Kernel;
use Derafu\Console\Runtime;

require dirname(__DIR__) . '/vendor/autoload.php';

exit(Runtime::run(fn (array $context): Kernel => new Kernel(
    $context['APP_ENV'],
    (bool) $context['APP_DEBUG'],
)));
```

A Project That Already Has Its Own Kernel

Use `ConsoleKernelTrait` directly on a dedicated subclass, so the main Kernel is not forced to carry console-specific wiring it may rarely need:

```
use Derafu\Console\ConsoleKernelTrait;

class ConsoleApplication extends Application // your project's own Kernel.
{
    use ConsoleKernelTrait;

    protected function configure(
        ContainerConfigurator $configurator,
        ContainerBuilder $container
    ): void {
        parent::configure($configurator, $container);
        $this->configureConsole($container);
    }
}
```

```
// bin/console
use App\ConsoleApplication;
use Derafu\Console\Runtime;

require dirname(__DIR__) . '/vendor/autoload.php';

exit(Runtime::run(fn (array $context): ConsoleApplication => new ConsoleApplication(
    $context['APP_ENV'],
    (bool) $context['APP_DEBUG'],
)));
```

The same applies if the console needs to share the exact same `services.yaml` as, e.g., an HTTP Kernel: extend that Kernel instead of `Derafu\Console\Kernel` and `use ConsoleKernelTrait` the same way — see Backbone Console for a real example of this (`libredte-lib-core-console`'s

`ConsoleApplication` extends its own business Kernel, not `Derafu\Console\Kernel`).

Overriding the Application Built by `run()`

`ConsoleKernelTrait::createConsoleApplication()` builds the underlying `Symfony\Component\Console\Application` — override it to customize the name/version, or to install a `CommandLoaderInterface` (as Backbone Console does):

```
use Symfony\Component\Console\Application as SymfonyConsoleApplication;
use Symfony\Component\DependencyInjection\ContainerInterface;

class ConsoleApplication extends Application
{
    use ConsoleKernelTrait {
        createConsoleApplication as private buildBaseConsoleApplication;
    }

    protected function createConsoleApplication(
        ContainerInterface $container
    ): SymfonyConsoleApplication {
        $application = $this->buildBaseConsoleApplication($container);

        // e.g. $application->setCommandLoader(...);

        return $application;
    }
}
```

The alias (`createConsoleApplication as private buildBaseConsoleApplication`) is required to call the trait's own implementation: a trait's methods are mixed directly into the class, not inherited through a parent class, so `parent::createConsoleApplication()` would not resolve.

Customizing the Application Name and Version

Set these as container parameters (e.g. in `services.yaml`) and they will be picked up automatically:

```
parameters:
    kernel.app_name: 'My CLI'
    kernel.app_version: '1.0.0'
```

Requirements

PHP 8.5+. Depends on `derafu/kernel` and `symfony/console`.

Last updated on 09/09/2026