Operations

Marking Real Operations: `#[Operation]`

Anonymous · 24/08/2026

Marking Real Operations: `#[Operation]`

A `Worker`'s public methods are ordinary PHP methods — but not all of them are business logic. Trait helpers (`JobsAwareTrait`/`HandlersAwareTrait`/`OptionsAwareTrait`) and `ServiceInterface` itself (`getId()`, `getName()`, `getDescription()`) contribute public methods too, and reflection alone cannot tell those apart from a worker's genuine capabilities. `#[Operation]` is the explicit signal that draws that line: tag a method with it, and any consumer — an allow-list policy, generated documentation, anything else that needs to enumerate what a worker can really do — has a reliable answer instead of a guess based on visibility alone.

```
use Derafu\Backbone\Attribute\Operation;

class InvoiceBuilderWorker extends AbstractWorker implements WorkerInterface
{
    #[Operation]
    public function build(string $number, int $amount): array
    {
        // ...
    }
}
```

That's the entire required usage: no arguments, just the tag. Everything else this attribute offers is optional.

Not a Service Attribute

`#[Package]`/`#[Component]`/`#[Worker]`/`#[Job]`/`#[Handler]`/`#[Strategy]` (see Service Lifecycle) all mark a *class* as a registrable service with its own identity, hierarchy and lifecycle — `TARGET_CLASS`, extending `AbstractServiceMetadata`. `#[Operation]` is deliberately different: `TARGET_METHOD`, and it does **not** extend `AbstractServiceMetadata` or implement `ServiceMetadataInterface`. An operation isn't a service of its own — it's a capability of a `Worker` that already exists, with no `id` or parent reference to manage, because the containing class's own service attribute already provides that context.

What It Can Add on Top of Reflection

Every property exists only to say something reflection or a PHPDoc block cannot say on its own — none of it is required, and the normal case is to set none of it at all:

```
#[Operation(
    name: 'Create a draft invoice',
    description: 'Builds a draft from the given data, without emitting it.',
    parameters: [
        'number' => ['example' => 'F-001'],
        'amount' => ['example' => 15000, 'description' => 'Amount in the smallest
currency unit.'],
    ],
    results: [
        'success' => ['description' => 'The created draft.', 'example' => ['id' =>
'DR-001']],
        MissingParameterException::class => ['description' => 'A required parameter
was not provided.'],
    ],
)]
public function build(string $number, int $amount): array
```

- **`name`/`description`** override the method's own PHPDoc summary/description. Leave both `null` (the default, and the expected normal case) to keep using PHPDoc — this is not a place to duplicate what a good docblock already says; it exists for the rarer case where the text written for PHP maintainers isn't the text an external consumer should see.
- **`parameters`** overrides or extends what reflection already knows about each parameter, keyed by parameter name. Only the keys given are applied — reflection's own `type`/`required`/`default` stay exactly as reflected unless a key explicitly overrides them:
  - `'example'` — a realistic sample value. Reflection has no way to produce one on its own.
  - `'type'`/`'description'` — for when reflection's own type isn't precise enough (a union type collapsed to a plain string, a bare `array` that actually has a real shape).
- **`results`** documents outcomes, keyed however the consumer identifies each one — `'success'`, or the fully-qualified class name of an exception the operation can throw, with whatever data that consumer finds useful under each key. This attribute does not define what a key means or what it's for; it just holds what's given. A key is never anything transport-specific like an HTTP status code — resolving a scenario to something transport-specific (a status code, for instance) is a decision for whichever consumer needs that, not for this attribute.

Who Reads It

`derafu/backbone` defines `#[Operation]` and stops there — it has no idea what, if anything, ever reads it. Nothing in this package depends on `derafu/backbone-dispatcher`, on purpose: a business library can tag its workers' operations without pulling in anything about how they'll eventually be dispatched.

The two real consumers today both live in `derafu/backbone-dispatcher`:

- **`TaggedOperationPolicy`** — the recommended policy for anything reachable from outside PHP — only allows dispatching what's tagged, closing off every trait helper and `ServiceInterface` method this attribute exists to distinguish from.
- **`Inspector`**, and through it `#[Operation]` in Backbone Dispatcher and `derafu/backbone-api`'s `Documenter` — merge the `name`/`description`/`parameters`/`results` given here on top of what reflection and PHPDoc already produced, to build generated documentation (an OpenAPI spec, for instance) that says more than reflection alone ever could.

Last updated on 24/08/2026