

Handling Errors

Handling Errors

Anonymous · 10/10/2026

Handling Errors

A failed operation raises `BackboneBridgeError` or one of its subclasses — never a raw `phpy` call failure, and never a PHP exception object:

```
from derafu_backbone_bridge import MissingParameterError

try:
    dispatcher.dispatch('example_package.example_component.example_worker::sum')
except MissingParameterError as e:
    print(e.php_class) #
"Derafu\BackboneDispatcher\Exception\MissingParameterException"
```

Every `BackboneBridgeError` carries the full `Problem` behind it (in `.problem`), not just a class name and a message:

```
except MissingParameterError as e:
    e.problem.detail          # The exception's own message.
    e.problem.instance       #
"example_package.example_component.example_worker::sum"
    e.problem.timestamp      # 1755500000.123456 – Unix epoch float, same as
`ExecutionMetadata.timestamp`.
    e.problem.throwable.php_class # Same as `e.php_class` below – a read-only
shortcut to this.
    e.problem.throwable.file
    e.problem.throwable.line
    e.problem.throwable.trace
    e.metadata               # ExecutionMetadata of the failed attempt, or
`None` – see below.
```

`Problem/SafeThrowable` mirror PHP's `ProblemDetailInterface/SafeThrowableInterface`, in `snake_case`. Unlike PHP's own `ProblemDetail::toArray()`, `.throwable` here is **always** populated, regardless of whether the underlying `SafeDispatcher` was booted with `debug=True` or `False`: that flag only gates what PHP embeds when *serializing* a problem for an untrusted HTTP consumer, and does not apply to this bridge — `phpy` is an in-process, same-machine, trusted boundary.

`.metadata` is `None` when the failure came from `GenericExplorer` rather than `GenericDispatcher` —

`SafeExplorerInterface` does not measure `ExecutionMetadata` (see [Backbone Dispatcher](#)), so there is honestly none to give in that case; a dispatch failure always has one, and its `.timestamp` is the exact same clock reading as `.problem.timestamp` — not two independent captures.

`.php_class` is a read-only shortcut to `.problem.throwable.php_class` — it exists for backward-compatible ergonomics, not stored data:

```
from derafu_backbone_bridge import BackboneBridgeError

try:
    dispatcher.dispatch('example_package.example_component.example_worker::fail')
except BackboneBridgeError as e:
    print(e.php_class) # "RuntimeException" — a plain, unmapped PHP exception.
```

The hierarchy mirrors `derafu/backbone` and `derafu/backbone-dispatcher`'s own exceptions one level deep (`ServiceNotFoundError/PackageNotFoundError/ComponentNotFoundError/...`, `ResolverError/InvalidParameterTypeError/...`), so callers can catch broadly or narrowly exactly as they would in PHP. Anything unmapped still raises `BackboneBridgeError`.

A specific library's own domain exceptions are registered by its own bridge, not by this package:

```
dispatcher.exceptions.register('App\\Exception\\SomeDomainException', SomeDomainError)
```

Separately, a failure while booting PHP itself (a missing autoloader, a broken dependency, a bootstrap class or method that doesn't exist) raises `BootstrapError` — never a raw php error either, but also never confused with an operation failure, since booting happens before any `SafeDispatcherInterface` exists to produce a `Problem` from.

Last updated on 10/10/2026