

Docs » Base for Applications Category » Backbone Bridge Python Project

Backbone Bridge Python Project

Call Any Backbone-Dispatcher Library From Python

Anonymous · 22/09/2026 · #python

Call Any Backbone-Dispatcher Library From Python

github derafu/backbone-bridge-python last commit september  CI passing code size 64.2 KiB

open issues 0 pypi v0.2.1 downloads 90/month

Generic Python bridge for any [derafu/backbone-dispatcher](#)-based PHP library, via [swoole/phpy](#) — without a caller ever having to know `phpy` exists.

Why

`derafu/backbone-dispatcher`'s `SafeDispatcherInterface` already turns any Backbone operation into something that never throws a raw PHP exception and always returns a serializable value — exactly the contract a foreign-language caller needs. `swoole/phpy` lets Python call into a real, embedded PHP interpreter in the same process. This package is the glue between the two: it boots a real `SafeDispatcherInterface` and exposes it as `dispatch(operation_id, **params)`, translating PHP exceptions into typed Python exceptions on the way back.

It knows nothing about any specific library. A specific PHP library's own bridge subclasses `GenericDispatcher` (and, for exploring a package tree, `GenericExplorer` — see [Exploring the Package Tree](#) below), pre-wiring how its own `SafeDispatcherInterface`/`SafeExplorerInterface` gets booted — this is exactly what this package's own test suite does, against a real, minimal PHP fixture (no mocks):

```
class ExampleDispatcher(GenericDispatcher):
    _BOOTSTRAP_CLASS = 'Derafu\\TestsBackboneBridgePython\\Fixture\\Bootstrap'

    def __init__(self, autoload_path=None):
        super().__init__(self._BOOTSTRAP_CLASS, autoload_path=autoload_path)
```

Installation

```
pip install derafu-backbone-bridge
```

phpy itself is **not** a normal pip dependency and never will be resolvable from PyPI: it requires a PHP build with `--enable-embed`, so it must already be present system-wide wherever this package runs — see [Docker with Python and Caddy](#) for a ready-made image with an optional `PHPY_ENABLED` build. Install this package into a virtualenv created with `--system-site-packages` so it can see the system phpy.

PyPI has an unrelated package also named ``phpy``

There is a long-abandoned, unrelated package on PyPI called `phpy` (“call legacy PHP functions from Python”, 2013). It is **not** `swoole/phpy`. This package deliberately does not declare `phpy` as a dependency at all, specifically so `pip install` never silently resolves that decoy instead of the real, system-provided one.

Usage

```
from derafu_backbone_bridge import GenericDispatcher

dispatcher = GenericDispatcher(
    'Derafu\\TestsBackboneBridgePython\\Fixture\\Bootstrap',
    autoload_path='/path/to/backbone-bridge-python/tests/php/vendor/autoload.php',
)

result = dispatcher.dispatch(
    'example_package.example_component.example_worker::sum',
    a=5, b=7,
)
result.value          # 12
result.data_type      # "integer" — `gettype()`/`get_class()` of `value`, before PHP
                      # serializes it.
result.metadata       # ExecutionMetadata — see below.
```

`operation_id` uses the same `"package.component.worker::operation"` format as `OperationRequest::fromId()` on the PHP side. Every keyword argument becomes a named operation parameter — `b` falls back to the operation’s own PHP default (10) when omitted, exactly as it would for a direct PHP caller.

`dispatch()` always returns an `OperationResult` — never the bare value — because the PHP side’s own `OperationResultInterface::getMetadata()` is never optional either: a successful dispatch always has execution statistics attached, so the Python side always has somewhere to put them.

```
@dataclass(frozen=True)
class OperationResult:
    value: Any
    metadata: ExecutionMetadata
```

```
data_type: str
```

`data_type` mirrors PHP's `OperationResultInterface::getDataType()` — never `None` here, unlike the PHP interface: a failed dispatch never reaches `OperationResult` at all (it raises instead, see [Handling Errors](#)), so there is no failure case for it to be `None` for.

`ExecutionMetadata` mirrors PHP's `ExecutionMetadataInterface` field for field, just in `snake_case`:

```
result.metadata.started_at      # "2026-01-20T10:00:00+00:00"
result.metadata.finished_at
result.metadata.timestamp      # 1755500000.123456 — same moment as
`finished_at`, as a Unix epoch float.
result.metadata.real_time      # Seconds, wall-clock — the "real" of `time`.
result.metadata.user_time      # Seconds of CPU in user mode — the "user" of
`time`.
result.metadata.system_time    # Seconds of CPU in kernel mode — the "sys" of
`time`.
result.metadata.memory_used    # Bytes, a delta — can be negative if the GC freed
more than this dispatch allocated.
result.metadata.peak_memory    # Bytes, the whole process's peak up to this
point.
result.metadata.pid
result.metadata.load_average_1min # Plus 5min/15min variants.
```

Assumes Linux/macOS, same as the PHP side: built on `getrusage()/sys_getloadavg()`, neither of which exists on Windows — no Windows support is offered.

Where the PHP autoloader lives

`GenericDispatcher` resolves the PHP autoloader to `phpy.include()` from, in order: an explicit `autoload_path` argument, or the `BACKBONE_DISPATCHER_AUTOLOAD` environment variable. Exactly one variable is used regardless of which library is being bridged, because in practice a given process only ever hosts one bridge:

```
export BACKBONE_DISPATCHER_AUTOLOAD=/path/to/backbone-bridge-
python/tests/php/vendor/autoload.php
```

```
dispatcher = ExampleDispatcher() # No path needed if the env var is set.
```

Neither of those is `phpy`-specific knowledge — they're just "where does my dependency live," the same kind of configuration any bridge would need regardless of the underlying interop mechanism.

Exploring the Package Tree: `GenericExplorer`

`GenericExplorer` mirrors `GenericDispatcher`'s boot mechanism, wrapping a real

`SafeExplorerInterface` instead of a `SafeDispatcherInterface`, with the same 10 methods `SafeExplorerInterface` has on the PHP side:

```
from derafu_backbone_bridge import GenericExplorer

explorer = GenericExplorer(
    'Derafu\\TestsBackboneBridgePython\\Fixture\\Bootstrap',
    bootstrap_method='bootExplorer',
    autoload_path='/path/to/backbone-bridge-python/tests/php/vendor/autoload.php',
)

explorer.get_packages()
explorer.get_components('example_package')
explorer.get_workers('example_package', 'example_component')
explorer.get_operations('example_package', 'example_component', 'example_worker')
explorer.get_package('example_package', with_components=True)
explorer.get_component('example_package', 'example_component', with_workers=True)
explorer.get_worker('example_package', 'example_component', 'example_worker',
with_operations=True)
explorer.get_operation('example_package', 'example_component', 'example_worker',
'sum')
explorer.describe('example_package.example_component.example_worker')
explorer.tree('example_package.example_component.example_worker')
```

Every method returns a plain dict/list (or scalar), same shape `derafu/backbone-dispatcher`'s `ExplorerInterface` produces, or raises a `BackboneBridgeError` — see [Handling Errors](#) below, `GenericExplorer` uses the exact same mapping. Unlike `GenericDispatcher.dispatch()`, none of these return an `OperationResult`: there is no `ExecutionMetadata` to attach, since `SafeExplorerInterface` does not measure one.

Handling Errors

A failed operation raises `BackboneBridgeError` or one of its subclasses — never a raw php 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 `.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 `phpy` error either, but also

never confused with an operation failure, since booting happens before any `SafeDispatcherInterface` exists to produce a `Problem` from.

Architecture

`phpy` is only ever imported by `GenericDispatcher`, `GenericExplorer`, and the internal `_phpy_conversions` module they both share for turning a live PHP object into a plain Python one — never by the tests, a specific library's own bridge, or the application consuming it. Building one specific library's bridge means subclassing `GenericDispatcher/GenericExplorer` with its `bootstrap_class`, and never touching `phpy` directly:

```
class GenericDispatcher:
    _AUTOLOAD_PATH_ENV = 'BACKBONE_DISPATCHER_AUTOLOAD'

    def __init__(
        self,
        bootstrap_class: str,
        bootstrap_method: str = 'boot',
        bootstrap_args: tuple = (),
        exception_registry: ExceptionRegistry | None = None,
        autoload_path: str | None = None,
    ) -> None: ...

    def dispatch(self, operation_id: str, **params) -> OperationResult: ...
```

`ExceptionRegistry` (the PHP-class-name → Python-exception mapping, via `raise_for(problem, metadata=None)`), and the `OperationResult/Problem/SafeThrowable/ExecutionMetadata` dataclasses themselves, are all separate, independent components that never touch `phpy` — plain, immutable data holders that can be built, tested and reasoned about with no PHP interpreter involved at all. `_phpy_conversions` is the one place that bridges the two worlds, turning a live PHP object into one of those.

Requirements

Python 3.14+. `phpy` itself requires PHP built with `--enable-embed`.

Last updated on 22/09/2026