

Usage

Usage

Anonymous · 10/10/2026

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.

Last updated on 10/10/2026