

Documentación

Bridge Python

Bridge Python

Anonymous · 08/10/2026

Tabla de contenidos

Bridge Python	3
<i>Introducción</i>	4
<i>Uso del Bridge Python</i>	6
<i>Arquitectura</i>	8

Documentación » Bridge Python

Bridge Python

Bridge Python

Anonymous · 08/10/2026

Bridge Python

Última actualización el 08/10/2026

Documentación » Bridge Python » Introducción

Introducción

Bridge Python para la Biblioteca PHP

Anonymous · 08/10/2026

Bridge Python para la Biblioteca PHP

 CI  pypi  downloads 

Bridge que expone, desde Python, las operaciones de `libredte/libredte-lib-core`, a través del `Dispatcher`, usando `derafu-backbone-bridge` como base. Repositorio: [libredte-lib-core-bridge-python](#).

Importante

Aunque se llame “bridge”, `phpy` no le habla a un proceso PHP aparte: embebe el motor de PHP dentro de tu propio proceso Python (es PHP compilado con `--enable-embed`, cargado como librería, no un servidor al otro lado de una red). Tu aplicación Python queda igual de sujeta a los términos de la [licencia AGPL](#) que si usaras la [Biblioteca PHP](#) o el [Dispatcher](#) directamente. Si buscas integrarte sin ese efecto, revisa los [Servicios Web \(API\)](#), la [Imagen Docker \(API\)](#) o la [Imagen Docker \(Consola\)](#).

Requerimientos

Antes de instalar, confirma que tienes:

- Python 3.14 o superior, compilado con soporte para `phpy`. La imagen de referencia con esto ya listo es [ghcr.io/derafu/docker-python3.14-caddy-server:phpy](#).
- Una instalación real (vía Composer) del [Dispatcher](#) accesible en disco.

Instalación

Se instala con pip:

```
pip install libredte-lib-core-bridge
```

Ejemplo

Un uso mínimo:

```
from libredte_lib_core_bridge import Dispatcher

# environment='prod', debug=False por defecto.
dispatcher = Dispatcher()

result = dispatcher.dispatch(
    'billing.identifier.caf_loader::load',
    xml=xml_base64,
)
```

`dispatch()` nunca deja pasar un error de PHP sin tipar: lanza una excepción Python (`derafu_backbone_bridge.BackboneBridgeError` o una de sus subclases) en vez de una excepción PHP cruda.

Nota

Revisa [Uso del Bridge Python](#) para cómo apuntarlo a tu instalación del Dispatcher y el manejo de errores.

Términos y condiciones de uso

Este proyecto es software libre y puede ser redistribuido y/o modificado bajo los términos de la Licencia Pública General Affero de GNU (AGPL), versión 3 o (a elección) cualquier versión posterior. Ver [COPYING](#).

Última actualización el 08/10/2026

Documentación » Bridge Python » Uso del Bridge Python

Uso del Bridge Python

Uso completo del Bridge Python

Anonymous · 08/10/2026

Uso del Bridge Python

Repasa cómo apuntar `Dispatcher` a tu instalación del `Dispatcher`, despachar operaciones y manejar los errores que puedan ocurrir.

Ubicar el Dispatcher

`Dispatcher` no sabe dónde vive el código PHP en disco. Eso se resuelve vía la variable de entorno `BACKBONE_DISPATCHER_AUTOLOAD` (apuntando al `vendor/autoload.php` de tu instalación del [Dispatcher](#)):

```
autoload=/ruta/al/dispatcher/vendor/autoload.php
export BACKBONE_DISPATCHER_AUTOLOAD="$autoload"
```

```
from libredte_lib_core_bridge import Dispatcher

dispatcher = Dispatcher()
```

O pasando `autoload_path` explícito al construirlo, sin necesidad de la variable de entorno:

```
autoload_path = '/ruta/al/dispatcher/vendor/autoload.php'

# El entorno puede ser 'prod' (default), 'dev' o 'test'.
dispatcher = Dispatcher(
    environment='prod',
    debug=False,
    autoload_path=autoload_path,
)
```

Despachar operaciones

Con el dispatcher ya listo, cada llamada indica la operación y sus parámetros como argumentos con nombre:

```
result = dispatcher.dispatch(
```

```
'billing.identifier.caf_loader::load',  
xml=xml_base64,  
)
```

El identificador de la operación sigue la forma "package.component.worker:operation", y los parámetros se pasan como argumentos con nombre. Revisa [Funcionalidades](#) de la biblioteca PHP para ver qué paquetes, componentes y workers (y por lo tanto qué operaciones) están disponibles.

Manejo de errores

`dispatch()` nunca deja pasar un error de PHP sin tipar: lanza una excepción Python (`derafu_backbone_bridge.BackboneBridgeError` o una de sus subclases, como `LibredteLibCoreError` para errores propios de `libredte-lib-core`) en vez de una excepción PHP cruda.

Última actualización el 08/10/2026

Documentación » Bridge Python » Arquitectura

Arquitectura

Cómo está armado este paquete

Anonymous · 08/10/2026

Arquitectura

Así está pre-cableado el bridge:

- `Dispatcher` es una subclase de `derafu_backbone_bridge.GenericDispatcher`, pre-cableada con el `Bootstrap::boot()` de `libredte-lib-core-dispatcher`.
- No importa `phpy` en ningún punto: todo el manejo de `phpy` vive dentro de `derafu-backbone-bridge`.

Desarrollo

Los tests de este paquete corren contra la biblioteca real, sin mocks:

```
make install-dev

# ruff + tests.
make check
```

Los tests requieren `phpy` (solo disponible dentro del contenedor `docker-python3.14-caddy-server` construido con `PHPY_ENABLED=true`) y una instalación real de `libredte-lib-core-dispatcher` (con sus propias dependencias de Composer) accesible en disco. No usan ningún fixture externo: generan un CAF real y válido en memoria vía el propio `caf_faker:create` de `libredte-lib-core`.

Última actualización el 08/10/2026