

Documentación

Dispatcher

Dispatcher

Anonymous · 08/10/2026

Tabla de contenidos

Dispatcher	3
<i>Introducción</i>	4
<i>Uso del Dispatcher</i>	6
<i>Arquitectura</i>	8
<i>Deserializadores</i>	9

Documentación » Dispatcher

Dispatcher

Dispatcher

Anonymous · 08/10/2026

Dispatcher

Última actualización el 08/10/2026

Documentación » Dispatcher » Introducción

Introducción

Dispatcher para la Biblioteca PHP

Anonymous · 08/10/2026

Dispatcher para la Biblioteca PHP

 CI passing license AGPL-3.0+ downloads 161

Conector que expone las operaciones de `libredte/libredte-lib-core` de forma segura y agnóstica al transporte, usando [derafu/backbone-dispatcher](#) como base. Pensado como la capa intermedia entre LibreDTE y un bridge hacia otro lenguaje, aunque no depende de ninguno en particular.

Repositorio: [libredte-lib-core-dispatcher](#).

Importante

El Dispatcher corre dentro de tu propio proceso PHP (lo llamas directo, no por red): tu software se convierte en un trabajo derivado y queda sujeto a los términos de la [licencia AGPL](#), igual que si usaras la [Biblioteca PHP](#) directamente. Si buscas integrarte sin ese efecto, revisa los [Servicios Web \(API\)](#), la [Imagen Docker \(API\)](#) o la [Imagen Docker \(Consola\)](#).

Instalación

Se instala como cualquier paquete de Composer:

```
composer require libredte/libredte-lib-core-dispatcher
```

Ejemplo

Un uso mínimo: arrancar el dispatcher, despachar una operación y revisar si tuvo éxito.

```
use libredte\lib\CoreDispatcher\Bootstrap;
use Derafu\BackboneDispatcher\ValueObject\OperationRequest;

// SafeDispatcherInterface.
$dispatcher = Bootstrap::boot();

$request = new OperationRequest(
    'billing',
    'identifier',
```

```
'caf_loader',
'load',
['xml' => base64_encode($cafXml)],
);

// Nunca lanza excepciones.
$result = $dispatcher->dispatch($request);

if ($result->isSuccess()) {
    // Ya serializado (array), listo para cruzar un límite de
    // proceso o lenguaje.
    $caf = $result->getValue();
} else {
    // ProblemDetail (RFC 7807-like).
    $problem = $result->getProblem();
}
```

Nota

Revisa [Uso del Dispatcher](#) para el detalle completo de `Bootstrap::boot()`, `OperationRequest` y el manejo de errores.

Términos y condiciones de uso

Licenciado bajo AGPL-3.0+. Disponible en el archivo [COPYING](#).

Última actualización el 08/10/2026

Documentación » Dispatcher » Uso del Dispatcher

Uso del Dispatcher

Uso completo del Dispatcher

Anonymous · 08/10/2026

Uso del Dispatcher

Repasa en detalle cómo arrancar el dispatcher, construir una solicitud y leer su resultado — el ejemplo de la introducción, explicado paso a paso.

Arrancar el dispatcher

`Bootstrap::boot()` es el único punto de entrada público del paquete. Todo el resto del cableado (Deserializers, la cadena `DirectDispatcher` → `TypedDispatcher` → `SafeDispatcher` de `derafu/backbone-dispatcher`) está declarado en su `config/services.yaml`, no en PHP.

```
use libredte\lib\CoreDispatcher\Bootstrap;

// El entorno puede ser 'prod' (default), 'dev' o 'test'.
$dispatcher = Bootstrap::boot(environment: 'prod', debug: false);
```

Construir la solicitud

Con los 4 identificadores por separado:

```
use Derafu\BackboneDispatcher\ValueObject\OperationRequest;

$request = new OperationRequest(
    'billing',
    'identifier',
    'caf_loader',
    'load',
    ['xml' => base64_encode($cafXml)],
);
```

O con el identificador `"package.component.worker:operation"` como un solo string — la forma natural para quien llama desde el otro lado de un límite de proceso o lenguaje, como el [Bridge Python](#):

```
$request = OperationRequest::fromId(
```

```
'billing.identifier.caf_loader::load',  
['xml' => base64_encode($cafXml)],  
);
```

Nota

Revisa [Funcionalidades](#) de la biblioteca PHP para ver qué paquetes, componentes y workers (y por lo tanto qué operaciones) están disponibles.

Despachar y leer el resultado

`dispatch()` nunca lanza excepciones: siempre revisa `isSuccess()` antes de usar el resultado.

```
$result = $dispatcher->dispatch($request);  
  
if ($result->isSuccess()) {  
    // Ya serializado (array), listo para cruzar un límite de  
    // proceso o lenguaje.  
    $caf = $result->getValue();  
} else {  
    // ProblemDetail (RFC 7807-like).  
    $problem = $result->getProblem();  
}
```

Parámetros anidados

Los parámetros que corresponden a objetos (CAF, certificados, emisor, receptor, mandatario, etc.) se pueden pasar como arreglos: el dispatcher los deserializa automáticamente al tipo esperado por la operación. Revisa [Deserializadores](#) para el detalle de qué interfaces están cubiertas.

Última actualización el 08/10/2026

Documentación » Dispatcher » Arquitectura

Arquitectura

Cómo está armado este paquete

Anonymous · 08/10/2026

Arquitectura

Así está resuelto el cableado interno del paquete:

- `Bootstrap::boot(string $environment = 'prod', bool $debug = false)` es el único punto de entrada público. Todo el resto del cableado (Deserializers, la cadena `DirectDispatcher` → `TypedDispatcher` → `SafeDispatcher` de [derafu/backbone-dispatcher](#)) está declarado en `config/services.yaml`, no en PHP.
- `config/services.yaml` importa el de `libredte-lib-core` y sobrescribe `libredte.lib.core.project_dir: kernel.project_dir` se resuelve contra el paquete que actúa como punto de entrada, y ese rol lo cumple este paquete, no `libredte-lib-core`.
- `Derafu\BackboneDispatcher\Contract\ObjectFactoryInterface` está registrado `lazy: true` porque `DocumentBagDeserializer/SiiRequestDeserializer` necesitan ese mismo registro (para deserializar sus campos anidados) y a la vez son entradas de su propio mapa de deserializers — una referencia circular que `lazy: true` rompe.

Desarrollo

Para modificar este paquete, instala sus dependencias y corre las validaciones antes de proponer un cambio:

```
composer install

# PHPUnit con cobertura.
composer tests

# Nivel 5.
composer phpstan

# php-cs-fixer (dry-run).
composer phpcs
```

Última actualización el 08/10/2026

Documentación » Dispatcher » Deserializadores

Deserializadores

Qué tipos cubren y cómo agregar uno nuevo

Anonymous · 08/10/2026

Deserializadores

Los 9 Deserializers de este paquete cubren únicamente interfaces/tipos de libredte-lib-core (o de paquetes Derafu genéricos que Core usa): `DocumentBagInterface`, `XmlDocumentInterface`, `CafInterface`, `CertificateInterface`, `EmisorInterface`, `ReceptorInterface`, `MandatarioInterface`, `DocumentEnvelopeInterface`, `SiiRequestInterface`.

Agregar soporte a una clase nueva

Sigue este orden para decidir qué construir:

1. Revisar si la clase ya tiene un `fromArray(array $data): self` estático. Si lo tiene, no hace falta escribir nada: `FromArrayDeserializer` (el `$fallback` del registro) ya la cubre automáticamente, sin registrar nada en `services.yaml`.
2. Si no lo tiene, ver si ya existe una Factory a la que delegar (ver `EmisorDeserializer`, `ReceptorDeserializer`, `MandatarioDeserializer`).
3. Si no hay Factory, construir el objeto directamente, delegando los campos que a su vez son objetos en el `ObjectFactoryInterface` inyectado (ver `DocumentBagDeserializer`).
4. Si la construcción no tiene una única forma fija, decidir según qué claves están presentes (ver `CertificateDeserializer`, que elige entre `loadFromData()/loadFromKeys()`).
5. Todo Deserializer empieza narrowing su entrada con `assertArray()/assertString()` (heredados de `Abstract\AbstractDeserializer`).

`DocumentBag` no tiene Deserializer propio para sus campos `document` y `documentType`: siempre reciben `null`. Agregar los suyos cuando se necesiten.

Última actualización el 08/10/2026