Deep Dive into Symfony's

Dependency Injection Component

Pimcore Community Dev Days 2025 | Alexander M. Turek

about:me

Software Developer

💙 php + Symfony + Doctrine 🧡🖤

❤️‍🔥 Open Source ❤️‍🔥

🤎 Legacy Code 🤎

💕 Father of Four 💕

Alexander M. Turek

derrabus

Follow | ♡ Sponsor

Developer. **@symfony** Core Team. **@doctrine** Core Team. Likes working with code that has aged like a good whisky.

easybill

Die cloudbasierte Rechnungssoftware für KMUs und Onlinehändler

Flexibilität

Modernes Dokumentenmanagement

zeichnet sich durch seine Flexibilität aus

Umfangreiche Funktionen

Manuell oder automatisiert

Bedienung

Einfach und intuitiv

Angebot

Support
+49 2154 897 01 - 20

E-Mail
support@easybill.de

Fax
+49 2154 897 01 - 29

easybill

Dokumente einfach, rechts- und datensicher online erstellen!

easybill GmbH • Düsselstr. 21 • 41564 Kaarst

Musterfirma GmbH
Herrn Max Mustermann
Teststraße 1
12345 Musterort

Datum **22.08.2022**
Kunde **10001**
Angebot **10007**

Sehr geehrter Herr Mustermann,

wie telefonisch besprochen, freuen wir uns Ihnen folgendes Angebot unterbreiten zu dürfen:

Angebot 10007

Pos	Bezeichnung	Menge	Einzelpreis	Betrag
1	Musterware	20	27,50	550,00 €
2	Musterware	10	32,80	328,00 €
3	Musterware	7	3,49	24,43 €

Nettobetrag	902,43 €
Umsatzsteuer 19%	171,46 €
Angebotsbetrag	**1.073,89 €**

Dieses Angebot ist gültig bis zum 05.09.2022.

Wenn es Ihnen zusagt, freuen wir uns über die Erteilung Ihres Auftrags. Dafür genügt ein Anruf unter folgender Telefonnummer: 01234 - 56789 - 10 oder eine kurze Mail an test@testfirma.de.
Selbstverständlich stehen wir Ihnen auch bei eventuellen Fragen zum Angebot gerne zur Verfügung.

Mit freundlichen Grüßen

Benjamin Klein
easybill GmbH

Verwaltung
Düsselstr. 21
41564 Kaarst
Support/Technik
Drahtzieherweg 9
47877 Willich

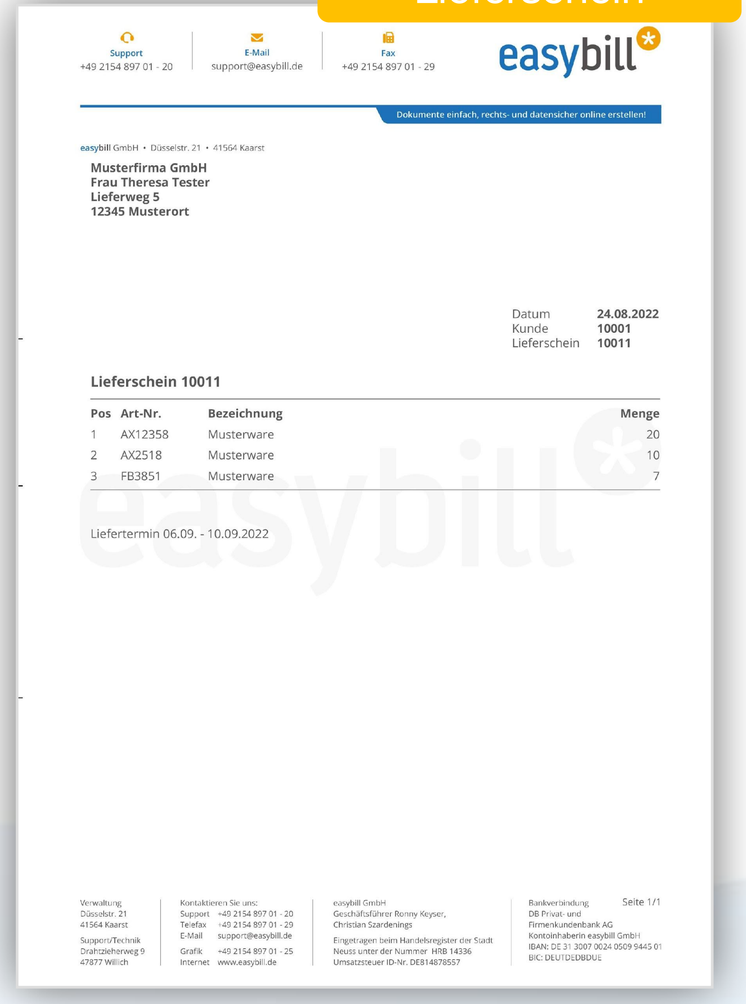
Kontaktieren Sie uns:
Support +49 2154 897 01 - 20
Telefax +49 2154 897 01 - 29
E-Mail support@easybill.de
Grafik +49 2154 897 01 - 25
Internet www.easybill.de

easybill GmbH
Geschäftsführer Ronny Keyser,
Christian Szardenings
Eingetragen beim Handelsregister der Stadt
Neuss unter der Nummer HRB 14336
Umsatzsteuer ID-Nr. DE814878557

Bankverbindung
DB Privat- und
Firmenkundenbank AG
Kontoinhaberin easybill GmbH
IBAN: DE 31 3007 0024 0509 9445 01
BIC: DEUTDEDBDUE

Seite 1/1

Lieferschein



Support
+49 2154 897 01 - 20

E-Mail
support@easybill.de

Fax
+49 2154 897 01 - 29

easybill

Dokumente einfach, rechts- und datensicher online erstellen!

easybill GmbH • Düsselstr. 21 • 41564 Kaarst

Musterfirma GmbH
Frau Theresa Tester
Lieferweg 5
12345 Musterort

Datum **24.08.2022**
Kunde **10001**
Lieferschein **10011**

Lieferschein 10011

Pos	Art-Nr.	Bezeichnung	Menge
1	AX12358	Musterware	20
2	AX2518	Musterware	10
3	FB3851	Musterware	7

Liefertermin 06.09. - 10.09.2022

Verwaltung
Düsselstr. 21
41564 Kaarst
Support/Technik
Drahtzieherweg 9
47877 Willich

Kontaktieren Sie uns:
Support +49 2154 897 01 - 20
Telefax +49 2154 897 01 - 29
E-Mail support@easybill.de
Grafik +49 2154 897 01 - 25
Internet www.easybill.de

easybill GmbH
Geschäftsführer Ronny Keyser,
Christian Szardenings
Eingetragen beim Handelsregister der Stadt
Neuss unter der Nummer HRB 14336
Umsatzsteuer ID-Nr. DE814878557

Bankverbindung
DB Privat- und
Firmenkundenbank AG
Kontoinhaberin easybill GmbH
IBAN: DE 31 3007 0024 0509 9445 01
BIC: DEUTDEDBDUE

Seite 1/1

Rechnung

Support
+49 2154 897 01 - 20

E-Mail
support@easybill.de

Fax
+49 2154 897 01 - 29

easybill

Dokumente einfach, rechts- und datensicher online erstellen!

easybill GmbH • Düsselstr. 21 • 41564 Kaarst

Musterfirma GmbH
Herrn Max Mustermann
Teststraße 1
12345 Musterort

Datum **01.09.2022**
Kunde **10001**
Rechnung **202210044**

Lieferanschrift:
Musterfirma GmbH
Frau Theresa Tester
Lieferweg 5
12345 MÜSTERORT

Rechnung 202210044
Das Rechnungsdatum entspricht dem Leistungsdatum

Pos	Bezeichnung	Menge	Einzelpreis	Betrag
1	Musterware	20	32,73	654,50 €
2	Musterware	10	39,03	390,32 €
3	Musterware	7	4,15	29,07 €

Nettobetrag	902,43 €
Umsatzsteuer 19%	171,46 €
Rechnungsbetrag	**1.073,89 €**

Vielen Dank für Ihren Auftrag!

Wir bitten um Überweisung des Rechnungsbetrages bis zum 15.09.2022 an die unten genannte Bankverbindung.

Mit freundlichen Grüßen

easybill GmbH - Buchhaltung

Verwaltung
Düsselstr. 21
41564 Kaarst
Support/Technik
Drahtzieherweg 9
47877 Willich

Kontaktieren Sie uns:
Support +49 2154 897 01 - 20
Telefax +49 2154 897 01 - 29
E-Mail support@easybill.de
Grafik +49 2154 897 01 - 25
Internet www.easybill.de

easybill GmbH
Geschäftsführer Ronny Keyser,
Christian Szardenings
Eingetragen beim Handelsregister der Stadt
Neuss unter der Nummer HRB 14336
Umsatzsteuer ID-Nr. DE814878557

Bankverbindung
DB Privat- und
Firmenkundenbank AG
Kontoinhaberin easybill GmbH
IBAN: DE 31 3007 0024 0509 9445 01
BIC: DEUTDEDBDUE

Seite 1/1

Symfony

a **framework** for PHP web applications
and
a set of reusable **components**

Symfony

Wrong Conference?

I'm not a Pimcore developer.

What is Dependency Injection?

```php
class NewsletterDispatcher
{
    public function send(array $recipients, string $subject, string $body): void
    {
        $mailer = new Mailer('smtp://myuser:p4ssw0rd@mail.example.com');
        $mailer->send($recipients, $subject, $body);
        $logger = new Logger('/var/log/app.log');
        $logger->log('Newsletter ' . $subject . ' successfully sent, sir!');
    }
}
```

Hard-wired Dependencies

Dependencies...

- ... are not changeable.
- ... are not configurable.
- ... will always be tested with your code.
- ... are not explicit.

```
class NewsletterDispatcher
{
    public function send(array $recipients, st
    {
        $mailer = new Mailer('smtp://myuser:p4
        $mailer->send($recipients, $subject, $
        $logger = new Logger('/var/log/app.log
        $logger->log('Newsletter ' . $subject
    }
}
```

What is Dependency Injection?

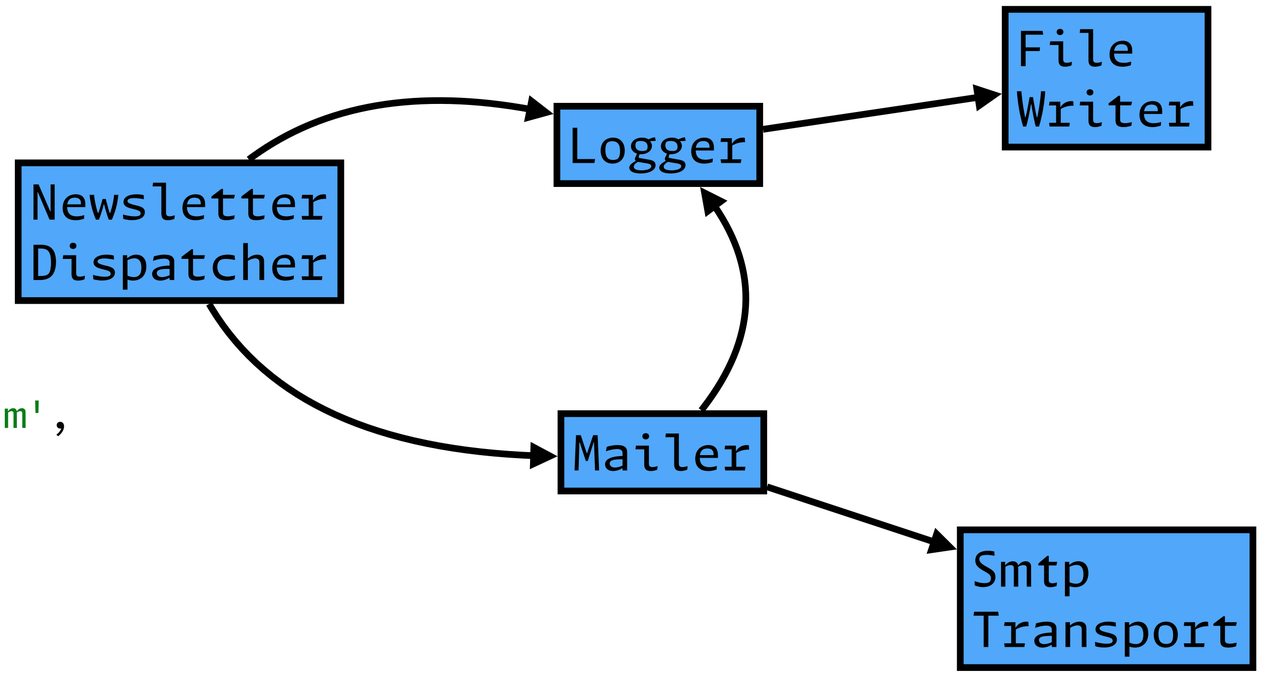
```php
final readonly class NewsletterDispatcher
{
    public function __construct(
        private Mailer $mailer,
        private Logger $logger,
    ) {
    }

    public function send(array $recipients, string $subject, string $body): void
    {
        $this->mailer->send($recipients, $subject, $body);
        $this->logger->log('Newsletter ' . $subject . ' successfully sent, sir!');
    }
}
```

What is Dependency Injection?

- A design pattern.
- It allows us to compose our application out of loosely coupled classes, so-called services.
- It enables us to develop against contracts instead of implementations (dependency inversion principle).

Dependency Graph

```
$logger = new Logger(
    new FileWriter(
        '/var/log/app.log',
    ),
);

$dispatcher = new NewsletterDispatcher(
    mailer: new Mailer(
        transport: new SmtpTransport(
            'smtp://myuser:p4ssw0rd@mail.example.com',
        ),
        logger: $logger,
    ),
    logger: $logger,
);
```



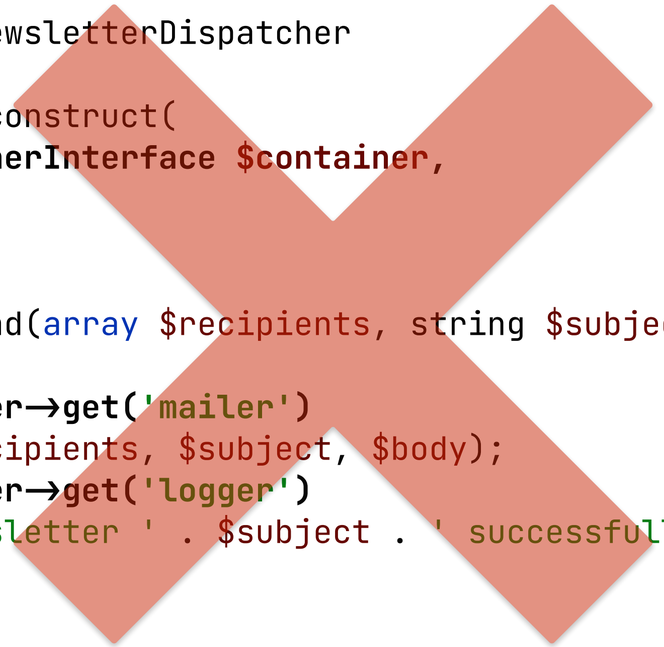
Service Location

- Another design pattern!
- A container for all of our services.
- The container…
  - … knows how to construct each service.
  - … remembers services after construction (singleton).
- Services are constructed recursively.

```
$dispatcher = $container
    ->get('newsletter_dispatcher');
```

Dependency Injection beats Service Location

```php
final readonly class NewsletterDispatcher
{
    public function __construct(
        private ContainerInterface $container,
    ) {
    }

    public function send(array $recipients, string $subject, string $body): void
    {
        $this->container->get('mailer')
            ->send($recipients, $subject, $body);
        $this->container->get('logger')
            ->log('Newsletter ' . $subject . ' successfully sent, sir!');
    }
}
```



Symfony Dependency Injection

A
PSR-11 compatible
dependency injection container

Symfony

services.yaml

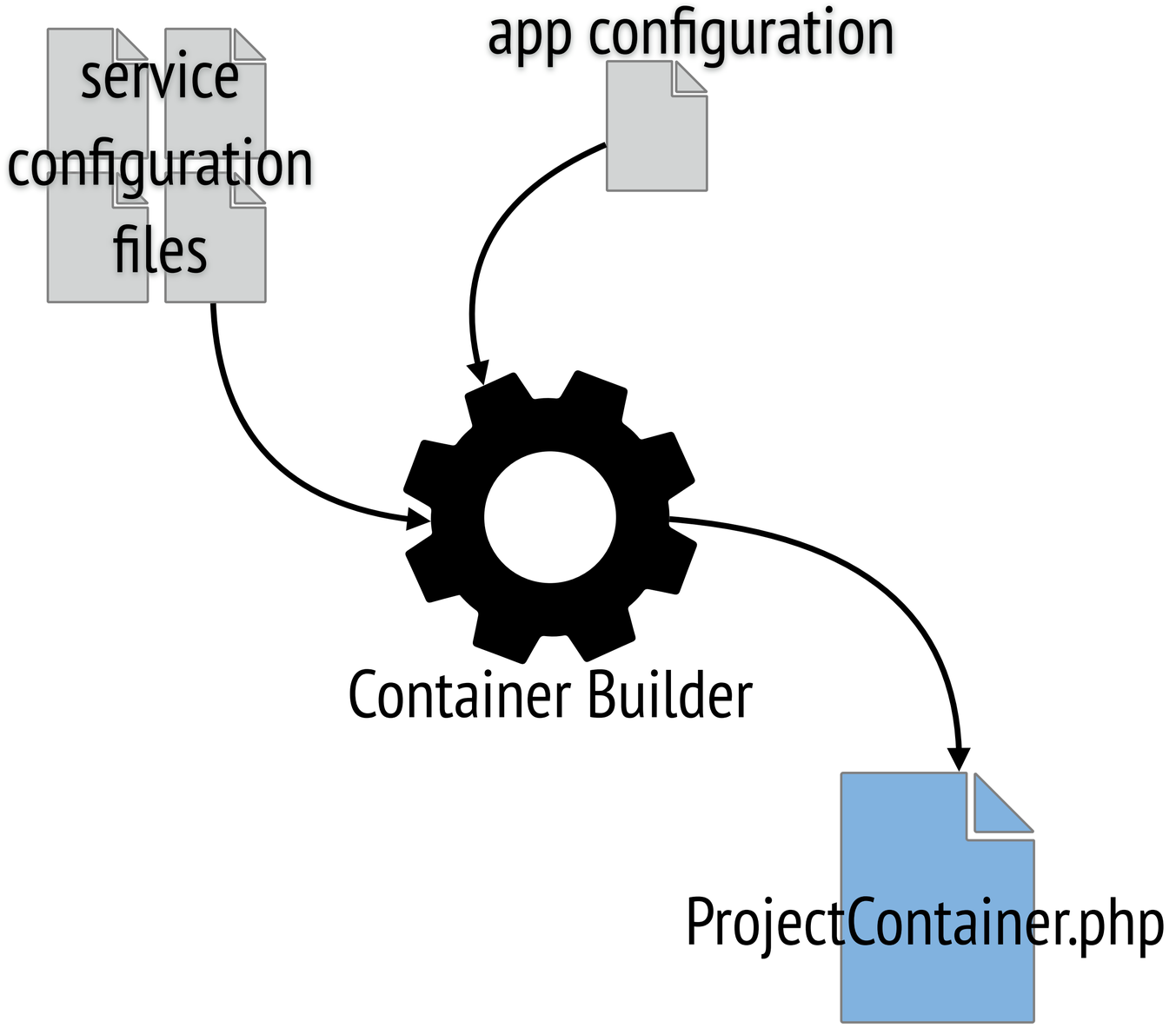
```yaml
services:
    file_writer:
        class: App\Logger\FileWriter
        arguments:
            $path: '/var/log/newsletter.log'

    logger:
        class: App\Logger\Logger
        arguments:
            $writer: '@file_writer'
```

```yaml
services:
    smtp_transport:
        class: App\Mailer\SmtpTransport
        arguments:
            $dsn: 'smtp://myuser:p4ssw0rd@mail.example.com'

    mailer:
        class: App\Mailer\Mailer
        arguments:
            $transport: '@smtp_transport'
            $logger: '@logger'

    newsletter_dispatcher:
        public: true
        class: App\Newsletter\NewsletterDispatcher
        arguments:
            $mailer: '@mailer'
            $logger: '@logger'
```

Container Compilation

The container is compiled to PHP.

No YAML parsing at runtime!
Container can be optimized.
We can transform the dependency tree during compilation.



Immutable Services

- Services are singletons (by default).
- Avoid stateful services.
- Don't inject data objects (domain models, DTOs).

Public and Private Services

Private Service

- **Can** be used as a dependency for other services.
- **Unavailable** through `$container->get($serviceId);`
- Might get inlined if only one dependent.
- Might be removed when unused.
- Services are **private by default**.

Public Service

- **Can** be used as a dependency for other services.
- **Can** be located through `$container->get($serviceId);`
- Won't be inlined or removed during compilation.

Our Services, Revisited.

```yaml
services:
    file_writer:
        class: App\Logger\FileWriter
        arguments:
            $path: '/var/log/newsletter.log'

    logger:
        class: App\Logger\Logger
        arguments:
            $writer: '@file_writer'
```

```yaml
services:
    smtp_transport:
        class: App\Mailer\SmtpTransport
        arguments:
            $dsn: 'smtp://myuser:p4ssw0rd@mail.example.com'

    mailer:
        class: App\Mailer\Mailer
        arguments:
            $transport: '@smtp_transport'
            $logger: '@logger'

    newsletter_dispatcher:
        public: true
        class: App\Newsletter\NewsletterDispatcher
        arguments:
            $mailer: '@mailer'
            $logger: '@logger'
```

Parameters

- Parameters can store scalar values or arrays.
- Load parameters from external sources,
  e.g. proprietary configuration files.
- Load different parameters for each environment.

```yaml
parameters:
    mailer_dsn: 'smtp://myuser:p4ssw0rd@mail.example.com'
    log_path: '/var/log/newsletter.log'

services:
    file_writer:
        class: App\Logger\FileWriter
        arguments:
            $path: '%log_path%'

    smtp_transport:
        class: App\Mailer\SmtpTransport
        arguments:
            $dsn: '%mailer_dsn%'

    # Other services remain unchanged
```

Environment Variables

- Load configuration from the environment.
- We (usually) don't need to recompile the container when the environment variables change!

```yaml
parameters:
    env(APP_LOG_PATH): '/var/log/newsletter.log'

services:
    file_writer:
        class: App\Logger\FileWriter
        arguments:
            $path: '%env(APP_LOG_PATH)%'

    smtp_transport:
        class: App\Mailer\SmtpTransport
        arguments:
            $dsn: '%env(APP_MAILER_DSN)%'

    # Other services remain unchanged
```

FQCN as Service IDs

Naming things is hard!

If we use the
fully-qualified class name (FQCN)
as service ID,
we can omit the class property.

```yaml
services:
    App\Logger\FileWriter:
        arguments:
            $path: '%log_path%'

    App\Logger\Logger:
        arguments:
            $writer: '@App\Logger\FileWriter'

    App\Mailer\SmtpTransport:
        arguments:
            $dsn: '%mailer_dsn%'

    App\Mailer\Mailer:
        arguments:
            $transport: '@App\Mailer\SmtpTransport'
            $logger: '@App\Logger\Logger'

    App\Newsletter\NewsletterDispatcher:
        public: true
        arguments:
            $mailer: '@App\Mailer\Mailer'
            $logger: '@App\Logger\Logger'
```

This Should Be Easier!

```yaml
App\Newsletter\NewsletterDispatcher:
    public: true
    arguments:
        $mailer: '@App\Mailer\Mailer'
        $logger: '@App\Logger\Logger'
```

```php
namespace App\Newsletter;

use App\Logger\Logger;
use App\Mailer\Mailer;

final readonly class NewsletterDispatcher
{
    public function __construct(
        private Mailer $mailer,
        private Logger $logger,
    ) {
    }

    // …
}
```

Autowiring

Allows us to under-specify
the list of constructor arguments.

Dependencies are looked up based on
parameter type-hints.

```yaml
services:
    App\Logger\FileWriter:
        arguments:
            $path: '%log_path%'

    App\Logger\Logger:
        arguments:
            $writer: '@App\Logger\FileWriter'

    App\Mailer\SmtpTransport:
        arguments:
            $dsn: '%mailer_dsn%'

    App\Mailer\Mailer:
        autowire: true
        arguments:
            $transport: '@App\Mailer\SmtpTransport'

    App\Newsletter\NewsletterDispatcher:
        public: true
        autowire: true
```

Service Aliases

Make a service available
under an addition service ID.

Allows us to autowire interfaces.

```yaml
services:
    App\Logger\FileWriter:
        arguments:
            $path: '%log_path%'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Logger\Logger:
        autowire: true

    App\Mailer\SmtpTransport:
        arguments:
            $dsn: '%mailer_dsn%'

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'

    App\Mailer\Mailer:
        autowire: true

    App\Newsletter\NewsletterDispatcher:
        public: true
        autowire: true
```

#[Autowire] Attribute

Allows us to control autowiring
for a specific constructor parameter.

```yaml
App\Mailer\SmtpTransport:
    autowire: true
```

```php
namespace App\Mailer;

use Symfony\Component\DependencyInjection\Attribute\Autowire;

final readonly class SmtpTransport implements TransportInterface
{
    public function __construct(
        #[Autowire(param: 'mailer_dsn')]
        private string $dsn,
    ) {
    }

    // …
}
```

#[Autowire] Attribute

Allows us to control autowiring
for a specific constructor parameter.

```php
#[Autowire(param: 'some_parameter')]
#[Autowire(env: 'SOME_VARIABLE')]
#[Autowire(service: 'my_service')]
#[Autowire('%kernel.project_dir%/var/log')]
```

Defaults

Allows us to apply settings
to all services defined
inside the **current configuration file**.

```yaml
services:
    _defaults:
        autowire: true

    App\Logger\FileWriter: ~

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Logger\Logger: ~

    App\Mailer\SmtpTransport: ~

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'

    App\Mailer\Mailer: ~

    App\Newsletter\NewsletterDispatcher:
        public: true
```

PSR-4 Autodiscovery

Because… who wants to enumerate all classes of their project anyway?

Remember: Unused private services are purged from the container on compilation.

```yaml
services:
    _defaults:
        autowire: true

    App\:
        resource: 'src/'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'

    App\Newsletter\NewsletterDispatcher:
        public: true
```

Autoconfiguration

Configure services automatically based on their type.

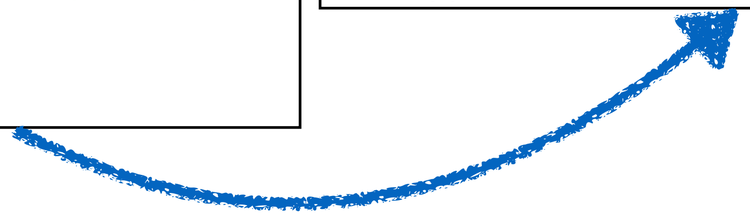
```yaml
App\Newsletter\NewsletterDispatcher:
    autoconfigure: true
```

```php
use App\Logger\Logger;
use App\Mailer\Mailer;
use Symfony\Component\DependencyInjection\Attribute\Autoconfigure;

#[Autoconfigure(public: true)]
final readonly class NewsletterDispatcher
{
    public function __construct(
        private Mailer $mailer,
        private Logger $logger,
    ) {
    }

    // …
}
```

Autoconfigure Everything

```yaml
services:
    _defaults:
        autowire: true


    App\:
        resource: 'src/'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'

    App\Newsletter\NewsletterDispatcher:
        autoconfigure: true
```

```yaml
services:
    _defaults:
        autowire: true
        autoconfigure: true

    App\:
        resource: 'src/'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'
```



```yaml
services:
    _defaults:
        autowire: true
        autoconfigure: true

    App\:
        resource: 'src/'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'

    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'
```

Task: A CSV Importer

- We want to import data from a CSV file.
- We only get strings from that file, but we want properly typed values.
- Data should be validated, bad rows must be rejected.
- We already have a component that gives us associative arrays.

```
id,last_name,first_name,alive
1,Shatner,William,yes
2,Nimoy,Leonard,no
3,Kelly,DeForest,no
4,Takei,George,yes
```

```
[
    'id' ⇒ '1',
    'last_name' ⇒ 'Shatner',
    'first_name' ⇒ 'William',
    'alive' ⇒ 'yes',
]
```

```
[
    'id' => 1,
    'last_name' => 'Shatner',
    'first_name' => 'William',
    'alive' => true,
]
```

```php
private function transformValue(string $key, string $value): mixed
{
    switch ($key) {
        case 'id':
            if (preg_match('/^[1-9]\d*$/', $value) !== 1) {
                throw new ImportFileMalformedException(sprintf('Invalid ID: %s', $value));
            }

            return (int) $value;

        // …

        case 'alive':
            return match ($value) {
                'yes' => true,
                'no' => false,
                default => throw new ImportFileMalformedException('Bad boolean value'),
            };
    }
}
```

```php
namespace App\Import\Transformer;

use App\Import\ImportFileMalformedException;

interface ColumnValueTransformerInterface
{
    public static function getColumnName(): string;

    /** @throws ImportFileMalformedException */
    public function transform(string $value): mixed;
}
```

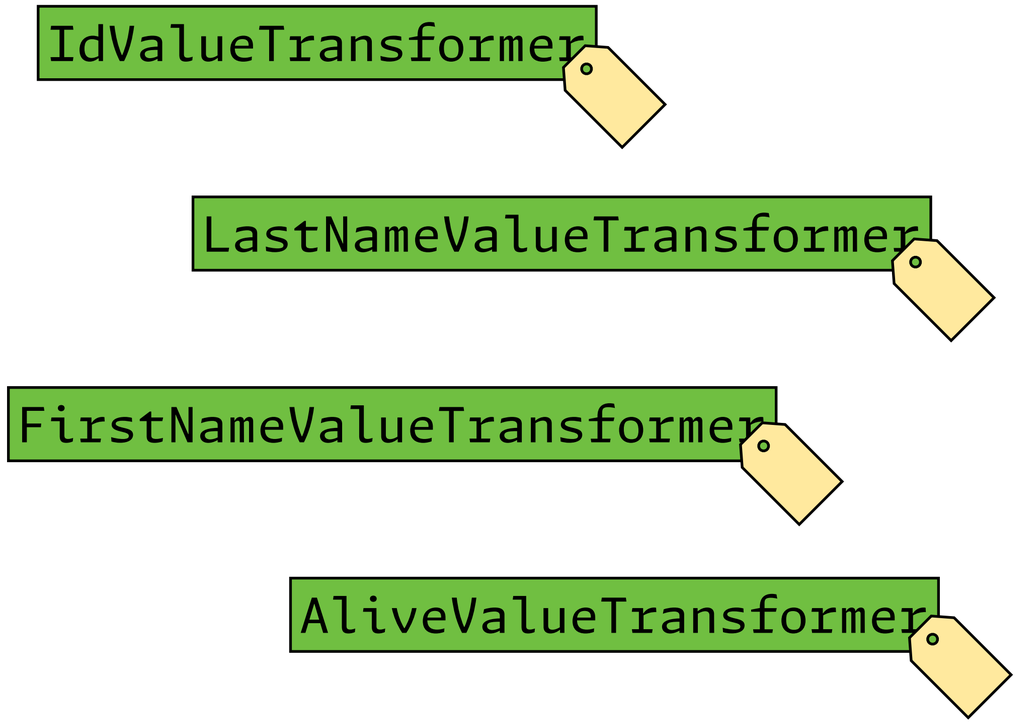
```php
namespace App\Import\Transformer;

use App\Import\ImportFileMalformedException;

final readonly class IdValueTransformer
    implements ColumnValueTransformerInterface
{
    public static function getColumnName(): string
    {
        return 'id';
    }

    public function transform(string $value): int
    {
        if (preg_match('/^[1-9]\d*$/', $value) !== 1) {
            throw new ImportFileMalformedException(
                sprintf('Invalid ID: %s', $value),
            );
        }

        return (int) $value;
    }
}
```

```php
namespace App\Import;

use App\Import\Transformer\ColumnValueTransformerInterface;

final readonly class FileImporter
{
    /** @param iterable<ColumnValueTransformerInterface> $columnValueTransformers */
    public function __construct(
        private iterable $columnValueTransformers,
    ) {
    }

    // …

    /** @throws ImportFileMalformedException */
    private function transformValue(string $key, string $value): mixed
    {
        foreach ($this->columnValueTransformers as $transformer) {
            if ($transformer->getColumnName() === $key) {
                return $transformer->transform($value);
            }
        }

        throw new \InvalidArgumentException(sprintf('Unknown key: %s', $key));
    }
}
```

```yaml
services:
    _defaults:
        autowire: true
        autoconfigure: true

    App\:
        resource: '../../src/'

    App\Import\FileImporter:
        arguments:
            $columnValueTransformers:
                - '@App\Import\Transformer\IdValueTransformer'
                - '@App\Import\Transformer\LastNameValueTransformer'
                - '@App\Import\Transformer\FirstNameValueTransformer'
                - '@App\Import\Transformer\AliveValueTransformer'

    App\Logger\LogWriterInterface: '@App\Logger\FileWriter'
    App\Mailer\TransportInterface: '@App\Mailer\SmtpTransport'
```

Tagged Services

- Services can have as many tags attached as we like.
- During compilation, we can easily find all services that have a specific tag attached.
- After compilation, the tags have no meaning.



#[AutoconfigureTag]
#[AutowireIterator]

Tag all services
that implement a specific interface.

Inject a collection of
all services with a given tag.

```php
namespace App\Import\Transformer;

use App\Import\ImportFileMalformedException;
use Symfony\Component\DependencyInjection\Attribute\AutoconfigureTag;

#[AutoconfigureTag('column_value_transformer')]
interface ColumnValueTransformerInterface
{
    // ...
}
```

```php
public function __construct(
    #[AutowireIterator('column_value_transformer')]
    private iterable $columnValueTransformers,
) {
}
```

```php
namespace App\Import;

use App\Import\Transformer\ColumnValueTransformerInterface;
use Psr\Container\ContainerInterface;
use Psr\Container\NotFoundExceptionInterface;
use Symfony\Component\DependencyInjection\Attribute\AutowireLocator;

final readonly class FileImporter
{
    public function __construct(
        #[AutowireLocator('column_value_transformer', defaultIndexMethod: 'getColumnName')]
        private ContainerInterface $columnValueTransformers,
    ) {
    }

    // …

    /** @throws ImportFileMalformedException */
    private function transformValue(string $key, string $value): mixed
    {
        try {
            $transformer = $this->columnValueTransformers->get($key);
        } catch (NotFoundExceptionInterface $e) {
            throw new \InvalidArgumentException(sprintf('Unknown key: %s', $key), previous: $e);
        }
        assert($transformer instanceof ColumnValueTransformerInterface);

        return $transformer->transform($value);
    }
}
```

Service Locators

Inject a **small container**
with a defined set of services.

A **projection** of the full service container.

Services are constructed
when we access them.

```php
namespace App\Import;

use App\Import\Transformer\ColumnValueTransformerInterface;
use Psr\Container\ContainerInterface;
use Psr\Container\NotFoundExceptionInterface;
use Symfony\Component\DependencyInjection\Attribute\AutowireLocator;

final readonly class FileImporter
{
    public function __construct(
        #[AutowireLocator('column_value_transformer', defaultIndexMethod:
        private ContainerInterface $columnValueTransformers,
    ) {
    }

    // …

    /** @throws ImportFileMalformedException */
    private function transformValue(string $key, string $value): mixed
    {
        try {
            $transformer = $this->columnValueTransformers->get($key);
        } catch (NotFoundExceptionInterface $e) {
            throw new \InvalidArgumentException(sprintf('Unknown key: %s',
        }
        assert($transformer instanceof ColumnValueTransformerInterface);

        return $transformer->transform($value);
    }
}
```

Problem: Log Calls to a HTTP Client

We want to trace when and how we make calls to external APIs.

Solution: A decorator.

```php
namespace App\HttpClient;

use App\Logger\Logger;
use Psr\Http\Client\ClientInterface;
use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;



final readonly class TraceableHttpClient implements ClientInterface
{
    public function __construct(

        private ClientInterface $httpClient,
        private Logger $logger,
    ) {}

    public function sendRequest(RequestInterface $request): ResponseInterface
    {
        $this->logger->log('Sending a request to ' . $request->getUri());

        return $this->httpClient->sendRequest($request);
    }
}
```

Decorators

Intercept calls to a given services using the popular decorator pattern.

```php
namespace App\HttpClient;

use App\Logger\Logger;
use Psr\Http\Client\ClientInterface;
use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;
use Symfony\Component\DependencyInjection\Attribute\AsDecorator;
use Symfony\Component\DependencyInjection\Attribute\AutowireDecorated;

#[AsDecorator(ClientInterface::class)]
final readonly class TraceableHttpClient implements ClientInterface
{
    public function __construct(
        #[AutowireDecorated]
        private ClientInterface $httpClient,
        private Logger $logger,
    ) {}

    public function sendRequest(RequestInterface $request): ResponseInterface
    {
        $this->logger->log('Sending a request to ' . $request->getUri());

        return $this->httpClient->sendRequest($request);
    }
}
```

Symfony Dependency Injection

- Pimcore leverages the Symfony DI Component.
- DI is a pattern for building loosely coupled architectures.
- Dependencies can be changed during tests or in developer mode.
- Autowiring makes configuring DI a breeze.

Symfony

Thank you!

Big thank you
to my sponsors!

💖

TYPO3

easybill

spam me:

me@derrabus.de

fund me:

https://github.com/sponsors/derrabus