**Sobre bundles en symfony 5**

**Following** [**https://symfonycasts.com/screencast/symfony-bundle/bundle-services#play**](https://symfonycasts.com/screencast/symfony-bundle/bundle-services#play)

En este tutorial lo primero que se hace es crear el bundle dentro pero fuera del proyecto, es decir, fuera de su carpeta SRC aventurando que en un futuro será totalmente independiente, en pasos posteriores lo separará del todo, pero de momento, para probar compatibilidad y cohesión lo hace de este modo. En el capitulo 9 empieza su separación.

Crear un proyecto básico en symfony 5

composer create-project symfony/website-skeleton bundles52 5.2.\*

composer require apache-pack

Crear un directorio lib/nbrdelbundle/src

Y meter ahí la clase en cuestión

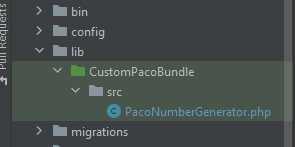
Como namespace ponerle el que quieras que sea el de tu bundle

namespace Paco\CustomPacoBundle;  
  
class PacoNumberGenerator  
{  
 public function getNumber(){  
 return 4564;  
 }  
}

Al intentar usar esta clase aunque en phpstorm te la linca y symfony te deja inyectarla obtendrás un class not found a la hora de ejecutarla =>

Add de añadir otra entrada en el psr-4 de composer.json

"autoload-dev": {  
 "psr-4": {  
 "Paco\\CustomPacoBundle\\": "lib/CustomPacoBundle/src/",  
 "App\\Tests\\": "tests/"  
 }  
},



Y ejecutar composer dump-autoload.

Ahora fallaría al intentar inyectarlo como servicio ya que el autowiring de symfony solo apunta a las clases dentro de src/ => voy al services.yml y añado mi clase disponible para autowiring añadiendo al final:

Paco\CustomPacoBundle\PacoNumberGenerator: ~

Ahora mismo tenemos una librería, que no es un bundle, la diferencia es que una librería exige que añadas configuración adicional para usarla como servicio en un contenedor de symfony mientras que un bundle nosotros añadimos esa configuración en el propio bundle y ya no es necesario añadir configuración extra fuera. Un bundle puede añadir automáticamente servicios al contenedor y para ello usaremos el directory DependencyInjection con su correspondiente clase, ver más abajo(2):

Creamos el archivo PacoNumberGeneratorBundle y lo añadimos al bundles.php

<?php  
  
  
namespace Paco\CustomPacoBundle;  
  
use Symfony\Component\HttpKernel\Bundle\Bundle;  
  
class PacoNumberGeneratorBundle extends Bundle  
{  
  
}

return [  
 Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],  
 Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],  
 Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],  
 Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],  
 Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],  
 Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true, 'test' => true],  
 Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],  
 Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],  
 Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],  
 Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],  
 Twig\Extra\TwigExtraBundle\TwigExtraBundle::class => ['all' => true],  
 Paco\CustomPacoBundle\PacoNumberGeneratorBundle::class => ['all' => true],  
];

(2) Creamos la carpeta DependencyInjection y dentro el archivo:

<?php  
  
  
namespace Paco\CustomPacoBundle\DependencyInjection;  
  
  
use Symfony\Component\DependencyInjection\ContainerBuilder;  
use Symfony\Component\DependencyInjection\Extension\Extension;  
  
class PacoNumberGeneratorExtension extends Extension  
{  
 public function load(array $configs, ContainerBuilder $container)  
 {  
 // *TODO: Implement load() method.* var\_dump("in the bundle");die;  
 }  
}

Cuando symfony intenta carga los bundles, itera sobre todos los bundles y busca por el directorio DependencyInjection, dentro por la clase con el mismo nombre del bundle terminada en “Extension”, luego si existe, la instancia y ejecuta el método “load” dentro de ella. Asi usaremos esto para añadir servicios… (..)

Ahora moveremos la notación que hicimos en el services.yml general a dentro de nuestro bunlde, asi borramos:

Paco\CustomPacoBundle\PacoNumberGenerator: ~

del services.yaml general y creamos el nuestro propio dentro de Resources/config/services.yaml:

*# config/services.yaml*services:  
 *# default configuration for services in \*this\* file* \_defaults:  
 autowire: true *# Automatically injects dependencies in your services.* autoconfigure: true *# Automatically registers your services as commands, event subscribers, etc.  
  
 # makes classes in src/ available to be used as services  
 # this creates a service per class whose id is the fully-qualified class name* Paco\CustomPacoBundle\:  
 resource: '../../../src/\*'  
 exclude: '../../../src/{DependencyInjection,Entity,Tests,Kernel.php}'  
  
 *# ...*

Y cargamos esa configuración dentro del método load del DependencyInjection:

<?php  
  
  
namespace Paco\CustomPacoBundle\DependencyInjection;  
  
  
use Symfony\Component\Config\FileLocator;  
use Symfony\Component\DependencyInjection\ContainerBuilder;  
use Symfony\Component\DependencyInjection\Extension\Extension;  
use Symfony\Component\DependencyInjection\Loader\YamlFileLoader;  
  
class PacoNumberGeneratorExtension extends Extension  
{  
 public function load(array $configs, ContainerBuilder $container)  
 {  
 // *TODO: Implement load() method.* $loader = new YamlFileLoader($container, new FileLocator(*\_\_DIR\_\_*.'/../Resources/config'));  
 $loader->load('services.yaml');  
 }  
}

Importante:

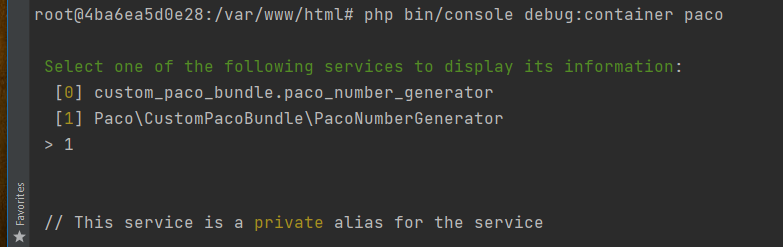
En nuestra aplicación usualmente llamaremos al id del servicio (alias) y no a su clase => vamos a añadir la configuración con el id del servicio a la declaración del mismo:

*# config/services.yaml*services:  
 *# default configuration for services in \*this\* file* \_defaults:  
 autowire: true *# Automatically injects dependencies in your services.* autoconfigure: true *# Automatically registers your services as commands, event subscribers, etc.  
  
 # makes classes in src/ available to be used as services  
 # this creates a service per class whose id is the fully-qualified class name  
# Paco\CustomPacoBundle\:  
# resource: '../../../src/\*'  
# exclude: '../../../src/{DependencyInjection,Entity,Tests,Kernel.php}'  
  
 # ...  
  
 # the id is not a class, so it won't be used for autowiring* custom\_paco\_bundle.paco\_number\_generator:  
 class: Paco\CustomPacoBundle\PacoNumberGenerator  
 public: true  
  
 *# but this fixes it!  
 # the ``app.rot13.transformer`` service will be injected when  
 # an ``App\Util\Rot13Transformer`` type-hint is detected* Paco\CustomPacoBundle\PacoNumberGenerator: '@custom\_paco\_bundle.paco\_number\_generator'

IMPOR:

*# the id is not a class, so it won't be used for autowiring*custom\_paco\_bundle.paco\_number\_generator:  
 class: Paco\CustomPacoBundle\PacoNumberGenerator  
 public: true  
  
*# but this fixes it!*Paco\CustomPacoBundle\PacoNumberGenerator: '@custom\_paco\_bundle.paco\_number\_generator'

Vemos como a pesar de figurar dos servicios uno es un alias del otro:



Los servicios pueden ser public o privados, por defecto son privados. Si se usan usando inyección de dependencias no es necesario que sean públicos, pero si alguien los quiere instanciar directamente si necesitará que estén seteados como public específicamente ( se supone…)

**Pasando argumentos a nuestro bundle desde la app principal**

Symfony procesa todos los yaml que hay en config packages y busca una Extension con la key a procesar a la que le pasará los datos, así creamos en config/packages:

Paco\_number\_generator.yaml:

paco\_number\_generator:  
 bar: true

y ese parámetro será pasado en el array de $configs a nuestro bundle

vale, ahora el tema está en pasarlo a la clase en cuestión de nuestro blunde, a su constructor

creamos la clase Configuration dentro de DependencyInjection y ahí checkeamos que los parámetros que nos pasan son del tipo valido:

class Configuration implements ConfigurationInterface  
{  
  
 public function getConfigTreeBuilder()  
 {  
 $treeBuilder = new TreeBuilder('custom\_paco\_bundle');  
 $rootNode = $treeBuilder->getRootNode();  
 $rootNode  
 ->children()  
 //->booleanNode('unicorns\_are\_real')->defaultTrue()->end()  
 ->integerNode('basenumber')->defaultValue(3)->end()  
 ->integerNode('topnumber')->defaultValue(3)->end()  
 ->end()  
 ;  
 return $treeBuilder;  
 }  
}

una vez hecho esto, dentro del PacoNumberGeneratorExtension en DependencyInjection recojo esa configuración y la paso a la clase:

public function load(array $configs, ContainerBuilder $container)  
{  
 //var\_dump($configs);die;  
 // *TODO: Implement load() method.* $loader = new YamlFileLoader($container, new FileLocator(*\_\_DIR\_\_*.'/../Resources/config'));  
 $loader->load('services.yaml');  
  
 //$loader = new XmlFileLoader($container, new FileLocator(\_\_DIR\_\_.'/../Resources/config'));  
 //$loader->load('services.xml');  
  
 //we pass the argument from the config/packages/paco\_number\_generator.yml to the class  
 $configuration = $this->getConfiguration($configs, $container);  
 $config = $this->processConfiguration($configuration, $configs);  
  
 $definition = $container->getDefinition('custom\_paco\_bundle.paco\_number\_generator');  
 $definition->setArgument(0, $config['basenumber']);  
 $definition->setArgument(1, $config['topnumber']);  
}

**Permitir inyectar o sobreescribir servicios de dentro del bundle desde fuera:**  
En el ejemplo haremos una clase provider que ponga una cadena aleatoria delante del numero y permitiremos desde fuera del bundle inyectar una clase de este tipo:

Añadimos una clase a nuestro bundle y la anexamos a nuestro constructor:

class MeetingMessageProvider  
{  
 public function getWordList(): array  
 {  
 return [  
 'Hi adorable ',  
 'Hi active ',  
 'Hi admire ',  
 'Hi adventurous ',  
 'Hi agreeable ',  
 ];  
 }  
}

class PacoNumberGenerator  
{  
 private $baseNumber;  
 private $topNumber;  
 private $meetingMessageProvider;  
  
 public function \_\_construct(MeetingMessageProvider $meetingMessageProvider,$baseNumber = 0, $topNumber = 100){  
 $this->baseNumber = $baseNumber;  
 $this->topNumber = $topNumber;  
 $this->meetingMessageProvider = $meetingMessageProvider;  
 }  
  
 public function getNumber(){  
 return random\_int($this->baseNumber, $this->topNumber);  
 }  
  
 private function getWordList(): array  
 {  
 return $this->meetingMessageProvider->getWordList();  
 }  
}

Tb la añadimos a nuestro services.yaml tal que :

If you're used to configuring services in YAML, the type="service" is equivalent to putting an @ symbol before the service id

*# config/services.yaml*services:  
 *# default configuration for services in \*this\* file* \_defaults:  
 autowire: true *# Automatically injects dependencies in your services.* autoconfigure: true *# Automatically registers your services as commands, event subscribers, etc.  
  
 # makes classes in src/ available to be used as services  
 # this creates a service per class whose id is the fully-qualified class name  
# Paco\CustomPacoBundle\:  
# resource: '../../../src/\*'  
# exclude: '../../../src/{DependencyInjection,Entity,Tests,Kernel.php}'* custom\_paco\_bundle.meeting\_message\_provider:  
 class: Paco\CustomPacoBundle\MeetingMessageProvider  
  
 *# the id is not a class, so it won't be used for autowiring* custom\_paco\_bundle.paco\_number\_generator:  
 class: Paco\CustomPacoBundle\PacoNumberGenerator  
 arguments: ['@custom\_paco\_bundle.meeting\_message\_provider']  
 public: true  
  
 *# but this fixes it!  
 # the ``app.rot13.transformer`` service will be injected when  
 # an ``App\Util\Rot13Transformer`` type-hint is detected* Paco\CustomPacoBundle\PacoNumberGenerator: '@custom\_paco\_bundle.paco\_number\_generator'

Y lo inyectamos via Configuration.php en DependencyInjection:

$rootNode  
 ->children()  
 //->booleanNode('unicorns\_are\_real')->defaultTrue()->end()  
 ->integerNode('basenumber')->defaultValue(3)->end()  
 ->integerNode('topnumber')->defaultValue(3)->end()  
 ->scalarNode('meeting\_message\_provider')->defaultNull()->end()  
 ->end()  
;

Y en PacoNumberGeneratorExtension dentro de DependencyInjection comprobamos que no nos pasan un servicio nulo y lo pasamos al constructor:

if (null !== $config['meeting\_message\_provider']) {  
 $definition->setArgument(0, $config['meeting\_message\_provider']);  
}  
  
$definition->setArgument(1, $config['basenumber']);  
$definition->setArgument(2, $config['topnumber']);

ok, hacemos algunas modificaciones en la clase general para que use ese provider y haga algo:

public function getNumber()  
{  
 $randomMeetingIndex = array\_rand($this->meetingMessageProvider->getWordList(), 1);  
 $randomNumber = random\_int($this->baseNumber, $this->topNumber);  
 $randomMeetingWord = $this->meetingMessageProvider->getWordList()[$randomMeetingIndex];  
 return $randomMeetingWord." ". $randomNumber;  
}

Ahora vamos a sobrecargar ese parámetro para que, desde la aplicación externa podamos pasar una clase custom al bundle:

Creamos un MeetingExternCustom tal que:

<?php  
  
  
namespace App\Services;  
  
  
use Paco\CustomPacoBundle\MeetingMessageProvider;  
  
class MeetingExternCustom extends MeetingMessageProvider  
{  
 public function getWordList(): array  
 {  
 //$words = parent::getWordList();  
 $words = ['Hi outside', 'Hi out'];  
 return $words;  
 }  
}

Y la inyectamos via paco\_number\_generator.yaml:

paco\_number\_generator:  
 basenumber: 200  
 topnumber: 205  
 meeting\_message\_provider: App\Services\MeetingExternCustom

Al refrescar nos da un petting:

**Argument 1 passed to Paco\CustomPacoBundle\PacoNumberGenerator::\_\_construct() must be an instance of Paco\CustomPacoBundle\MeetingMessageProvider, string given, called in /var/www/html/var/cache/dev/Container1GyvuK0/getCustomPacoBundle\_PacoNumberGeneratorService.php on line 22**

La razón del petting es que realmente se esta pasando una cadena y no esta construyendo la referencia al servicio externo como clase:

|  |
| --- |
| Go back to our extension class. Here's the fix: when we set the argument to $config['word\_provider'], this of course sets that argument to the string value! To fix this in YAML, we would prefix the service id with the @ symbol. In PHP, wrap the value in a new Reference() object. This tells Symfony that we're referring to a service. |

Para solventar eso hacemos esta modificación en el método load del PacoNumberGeneratorExtension.php dentro del dependencyInjection folder:

if (null !== $config['meeting\_message\_provider']) {  
 //$definition->setArgument(0, $config['meeting\_message\_provider']);  
 $definition->setArgument(0, new Reference($config['meeting\_message\_provider']));  
}

y todo ok, estamos cargado con un servicio externo un servicio usado dentro de nuestro bundle!

Paso de refinamiento, en este bloque lo que hacemos es dar un alias a ese servicio que hemos hecho al MeetingMessageProvider con lo que cambiaremos la forma en la que se setea como argumento y además crearemos un interface para que desde fuera el que quiera sobrecargar nuestra funcionalidad no tenga que extender de nuestra clase, solo tenga que implementar una interfaz.

Allá vamos!:

Pongo un “alias” al servicio y public =true

custom\_paco\_bundle.meeting\_message\_provider:  
 class: Paco\CustomPacoBundle\MeetingMessageProvider  
 public: true  
  
Paco\CustomPacoBundle\MeetingMessageProvider: '@custom\_paco\_bundle.meeting\_message\_provider'

Gracias a eso podemos eliminar la línea de NewReference cuando inyectamos las dependencias, quedaría asi:

if (null !== $config['meeting\_message\_provider']) {  
 //$definition->setArgument(0, $config['meeting\_message\_provider']);  
 //$definition->setArgument(0, new Reference($config['meeting\_message\_provider']));  
 $container->setAlias('custom\_paco\_bundle.meeting\_message\_provider', $config['meeting\_message\_provider']);  
}

Ahora creamos una interfaz para evitar que quien quiera implementar o personalizar un servicio dentro de nuestro bundle tenga que extender de una clase en lugar de implementar una interfaz:

namespace Paco\CustomPacoBundle\Interfaces;  
  
interface MeetingMessageProviderInterface  
{  
 */\*\*  
 \* Return an array of words to use for the fake text.  
 \*  
 \** ***@return*** *array  
 \*/* public function getWordList(): array;  
}

asi nuestra clase number generator recibirá esa clase como parámetro:

public function \_\_construct(MeetingMessageProviderInterface $meetingMessageProvider, $baseNumber = 0, $topNumber = 100)  
{

y tanto nuestro meetingMessageProvider como el servicio externo que hace lo mismo implementarán esa interfaz:

class MeetingMessageProvider implements MeetingMessageProviderInterface  
{

**9.- Separar el bundle, crear su propio composer.json**