Introducción

Los Bundles en Symfony se habilitan por entorno en el fichero config/bundles.php. Es decir, al habilitar cada bundle se debe indicar para qué entorno o entornos se va a habilitar:

```
// config/bundles.php
return [
    // 'all' means that the bundle is enabled for any Symf
ony environment
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class
=> ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class =>
 ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all'
=> true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => [
'all' => true],
    Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle::cl
ass => ['all' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class =
> ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtr
aBundle::class => ['all' => true],
    // this bundle is enabled only in 'dev'  and 'test', s
o you can't use it in 'prod'
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::cl
```

```
ass => ['dev' => true, 'test' => true],
];
```

Si utilizamos Flex, los bundles se habilitan y deshabilitan automáticamente, por lo que no necesitaremos editar el archivo bundles.php.

Creación de un Bundle

Vamos a crear un Bundle llamado TrainingitEjemploBundle.

Lo primero es crear el directorio **src/Trainingit/EjemploBundle/** y añadir un fichero llamado **TrainingitEjemploBundle.php**:

```
// src/Trainingit/EjemploBundle/TrainingitEjemploBundle.ph
p
namespace App\Trainingit\EjemploBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class TrainingitEjemploBundle extends Bundle
{
}
```

Solamente con este fichero, nuestro bundle ya es válido y se podría incluir en nuestra aplicación añadiéndolo en el fichero **config/bundles.php**:

```
// config/bundles.php
return [
    // ...
    App\Trainingit\EjemploBundle\TrainingitEjemploBundle::
class => ['all' => true],
];
```

Estructura de directorios de un Bundle

La estructura de directorio de un bundle es simple y flexible. Por defecto, un bundle sigue una serie de convenciones que ayuda a mantener una consistencia de unos bundles a otros.

- Controller/

Contiene los controladores

- Resources/config/

Contiene la configuración, incluidas las rutas

- Resources/views/

Contiene las plantillas twig

- Resources/public/

Contiene los assets

- Tests/

Contiene los Tests

Un bundle puede ser tan grande o tan pequeño como la funcionalidad que implemente. Solamente debe contener los ficheros y directorios que necesite y nada más.

Cómo sobrescribir partes de un Bundle

Muchas veces queremos sobrescribir partes de un bundle. A continuación se describe cómo sobrescribir los distintos elementos de un bundle.

Templates

Lo vimos en el tema 5 de Twig.

Imaginemos que queremos sobrescribir la plantilla **Resources/views/Blog/index.html.twig** del bundle **UnBundle**.

Para hacerlo, simplemente debemos copiar dicho archivo en **templates/bundles/UnBundle/Blog/index.html.twig** y realizar los cambios que deseemos en este archivo.

Después de esto, debemos limpiar la caché de symfony incluso si estamos trabajando en el entorno de desarrollo.

Routing

Los ficheros de routing de un bundle no son operativos. Es decir, que al instalar un bundle con rutas, debemos copiar de alguna forma esas rutas en nuestra configuración.

Para sobrescribir una ruta de un bundle simplemente bastará con modificarla cuando la copiemos a nuestra configuración

Controllers

Para sobrescribir un controlador, la técnica es crear una ruta + controlador nuevos con el mismo path del controlador que queremos sobrescribir.

Si la nueva ruta se carga antes que la del bundle, el controlador que se ejecutará será el nuestro.

Services & Configuration

Si queremos modificar la definición de algún servicio de un bundle, podemos utilizar un *compiler pass* para cambiar la clase asociada al servicio.

En el siguiente ejemplo, se modifica la clase asociada al servicio 'original-service-id' para que utilice la clase App\YourService:

```
// src/Kernel.php
namespace App;

// ...
+ use App\Service\YourService;
+ use Symfony\Component\DependencyInjection\ContainerBuild
```

```
er;
+ use Symfony\Component\DependencyInjection\Compiler\Compi
lerPassInterface;

class Kernel extends BaseKernel implements CompilerPassInt
erface
{
+     public function process(ContainerBuilder $container)
+     {
+         $definition = $container->findDefinition('origin
al-service-id');
+         $definition->setClass(YourService::class);
+     }
}
```

Entities & Entity Mapping

Debido a la forma en la que Doctrine trabaja, NO es posible sobrescribir el mapeo de las entidades. Si un bundle crea una superclase (como la entidad User de FOSUserBundle) entonces sí podemos sobrescribir atributos y sus mapeos extendiendo dicha clase.

Forms

Extendiendo las clases de formulario.

Translations

Las traducciones no están relacionadas con bundles, sino con *domains*. Esto significa que podemos sobrescribir traducciones de cualquier fichero de un bundle simplemente incorporando la traducción en un fichero con el dominio correcto en nuestra aplicación.

Creación de recipes

Las Recipes se definen en un archivo manifest.json y pueden contener cualquier número de otros ficheros y directorios. Las recipes se almacenan en su propio repositorio, fuera del repositorio del package.

Por ejemplo, este es el contenido del archivo manifest.json de SwiftmailerBundle:

```
{
    "bundles": {
        "Symfony\\Bundle\\SwiftmailerBundle\\SwiftmailerBu
ndle": ["all"]
    },
    "copy-from-recipe": {
        "config/": "%CONFIG_DIR%/"
    },
    "env": {
        "MAILER_URL": "smtp://localhost:25?encryption=&aut
h_mode="
    },
    "aliases": ["mailer", "mail"]
}
```

Todo el contenido del fichero manifest.json es opcional. Vamos a ver todas las opciones y configuradores que puede haber en el fichero.

Options

aliases option

Un array de strings.

Solamente disponible en el repositorio de recipes oficial. Define uno o más alias para instalar el paquete.

```
{
    "aliases": ["mailer", "mail"]
}
```

Los desarrolladores pueden utilizar cualquiera de los alias para instalar el paquete con Flex.

Configurators

Sirven para configurar las diferentes tareas que se deben ejecutar para instalar la dependencia, tales como ejecutar comandos, copiar ficheros, añadir variables de entorno…

Symfony es capaz de realizar los procesos inversos para desinstalar la dependencia.

bundles Configurator

Habilita uno o más bundles en la aplicación Symfony añadiéndolos al fichero bundles.php.

Su valor es un array asociativo en el que la clave es el nombre de la clase del bundle y el valor es un array con los entornos en los que debe ser habilitado.

Los entornos soportados son *dev*, *prod*, *test* y *all*.

```
{
    "bundles": {
        "Symfony\\Bundle\\DebugBundle\\DebugBundle": ["dev
", "test"],
        "Symfony\\Bundle\\MonologBundle\\MonologBundle": [
"all"]
    }
}
```

La anterior recipe se transformaría en el siguiente código:

```
// config/bundles.php
return [
    'Symfony\Bundle\DebugBundle\DebugBundle' => ['dev' =>
true, 'test' => true],
    'Symfony\Bundle\MonologBundle\MonologBundle' => ['all'
```

```
 => true],
];
```

container Configurator

Añade parámetros en el services.yaml.

```
{
    "container": {
        "locale": "en"
    }
}
```

copy-from-package Configurator

Copia ficheros o directorios del package de composer a la aplicación Symfony.

```
{
    "copy-from-package": {
        "bin/check.php": "%BIN_DIR%/check.php"
    }
}
```

Las variables especiales disponibles son: %BIN_DIR%, %CONF_DIR%, %CONFIG_DIR%, %SRC_DIR% %VAR_DIR% y %PUBLIC_DIR%.

También podemos acceder a variables que hayamos definido en la sección **extra** del fichero *composer.json*:

```
// composer.json
{
    "...": "...",

    "extra": {
        "my-special-dir": "..."
    }
}
```

En este caso tenemos disponible %MY_SPECIAL_DIR% un nuestras recipes.

copy-from-recipe Configurator

Igual que copy-from-package pero los contenidos se copian de la recipe directamente. Se puede utilizar para crear una estructura o configuración inicial.

```
"copy-from-recipe": {
    "config/": "%CONFIG_DIR%/",
    "src/": "%SRC_DIR%/"
}
```

env Configurator

Añade variables de entorno a nuestros ficheros .env y .env.dist

```
{
    "env": {
        "APP_ENV": "dev"
    }
}
```

Esta recipe se convierte en el sigiente código:

```
###> your-recipe-name-here ###
APP_ENV=dev
###< your-recipe-name-here ###
```

Los delimitadores son necesarios para que Symfony pueda eliminar las variables de entorno al desinstalar un paquete. No se deben quitar ni modificar dichos delimitadores.

composer-scripts Configurator

Scripts que se deben ejecutar automáticamente al ejecutar composer install y composer update.

Los tipos de scripts que se pueden ejecutar son php-script, script y

symfony-cmd.

```
{
    "composer-scripts": {
        "vendor/bin/security-checker security:check": "php
-script",
        "make cache-warmup": "script",
        "assets:install --symlink --relative %PUBLIC_DIR%"
: "symfony-cmd"
    }
}
```

gitignore Configurator

Añade patrones al fichero .gitignore

```
{
    "gitignore": [
        ".env",
        "/public/bundles/",
        "/var/",
        "/vendor/"
    ]
}
```

El contenido añadido al fichero .gitignore también se añade con delimitadores que no se deben quitar ni modificar.

post-install-output Configurator

Muestra contenido en la consola después de instalar el paquete.

El contenido debe definirse en un fichero llamado post-install.txt.

```
<bg=blue;fg=white>                </>
<bg=blue;fg=white> What's next? </>
<bg=blue;fg=white>                </>

  * <fg=blue>Run</> your application:
    1. Change to the project directory
    2. Execute the <comment>make serve</> command;
    3. Browse to the <comment>http://localhost:8000/</> UR
L.

  * <fg=blue>Read</> the documentation at <comment>https:/
/symfony.com/doc</>
```

Validation

Al enviar una recipe, se comprueban automáticamente las siguientes características:

- YAML files suffix must be .yaml, not .yml;
- YAML files must be valid;
- YAML files must use 4 space indentations;

- YAML files under config/packages must not define a “parameters” section;
- JSON files must be valid;
- JSON files must use 4 space indentations;
- Aliases are only supported in the main repository, not the contrib one;
- Aliases must not be already defined by another package;
- The manifest file only contains supported keys;
- The package must exist on Packagist;
- The package must have at least one version on Packagist;
- The package must have an MIT or BSD license;
- The package must be of type “symfony-bundle” if a bundle is registered in the manifest;
- The package must have a registered bundle in the manifest if type is “symfony-bundle”;
- The package does not only register a bundle for all environments;
- The package does not depend on symfony/symfony or symfony/security;
- All text files should end with a newline;
- All configuration file names under config should use the underscore notation;
- No “semantically” empty configuration files are created under config/packages;
- All files are stored under a directory referenced by the “copy-from-recipe” section of “manifest.json”;
- The pull request does not contain merge commits;
- The Symfony website must be referenced using HTTPs.

Ejemplo real

A modo de ejemplo, la siguiente recipe es la recipe de bundle FrameworkBundle:

```
{
    "bundles": {
        "Symfony\\Bundle\\FrameworkBundle\\FrameworkBundle
": ["all"]
    },
    "copy-from-recipe": {
        "config/": "%CONFIG_DIR%/",
        "public/": "%PUBLIC_DIR%/",
        "src/": "%SRC_DIR%/"
    },
    "composer-scripts": {
        "cache:clear": "symfony-cmd",
        "assets:install --symlink --relative %PUBLIC_DIR%"
: "symfony-cmd"
    },
    "env": {
        "APP_ENV": "dev",
        "APP_SECRET": "%generate(secret)%"
    },
    "gitignore": [
        ".env",
        "/public/bundles/",
```

```
        "/var/",
        "/vendor/"
    ]
}
```