Constraints en Symfony

Las constraints son restricciones o reglas a las que han de someterse los valores introducidos en una aplicación Symfony

Contenido modificable

Si ves errores o quieres modificar/añadir contenidos, puedes crear un pull request. Gracias

El validator service está diseñado para validar objetos frente a constraints (restricciones). Una constraint no es más que un objeto PHP que hace sentencias asertivas, y Symfony se asegura de que sean true. Dado un valor, un constraint te dirá si ese valor se adhiere a las normas de ese constraint.

Symfony viene por defecto con buen número de constraints que cubren la mayoría de necesidades, divididos en 9 secciones:

1. Basic Constraints	NotBlank, Blank, NotNull, IsNull, IsTrue, IsFalse, Type
2. String Constraints	Email, Length, Url, Regex, Ip, Uuid
3. Number Constraints	Range
4. Comparison Constraints	EqualTo, NotEqualTo, IdenticalTo, NotIdenticalTo, LessThan, LessThanOrEqual, GreaterThan, GreaterThanOrEqual
5. Date Constraints	Date, DateTime, Time
6. Collection Constraints	Choice, Collection, Count, UniqueEntity, Language, Locale, Country
7. File Constraints	File, Image
8. Financial and other Number Constraints	Bic, CardScheme, Currency, Luhn, Iban, Isbn, Issn
9. Other Constraints	Callback, Expression, All, UserPassword, Valid
10. Configuración opcional	

1. Basic Constraints

Valida que un valor no este vacío, no sea false, no sea igual a un string vacío ni sea null.

Aplicación: propiedades o métodos Opciones: message, payload Clase: NotBlank Validator: NotBlankValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\NotBlank()
     */
    protected $firstName;
}
```

Blank

Valida que un valor está vacío, ya sea un string vacío o igual a null.

Aplicación: propiedades o métodos Opciones: message, payload Clase: Blank Validator: BlankValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Blank()
     */
    protected $firstName;
}
```

NotNull

Valida que un valor no sea estrictamente igual que null.

Aplicación: propiedades o métodos Opciones: message, payload Clase: NotNull Validator: NotNullValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\NotNull()
     */
    protected $firstName;
}
```

Valida que un valor es exactamente igual a null.

Aplicación: propiedades o métodos Opciones: message, payload Clase: IsNull Validator: IsNullValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\IsNull()
     */
    protected $firstName;
}
```

IsTrue

Valida que un valor sea true, 1 o "1".

Aplicación: propiedades o métodos Opciones: message, payload Clase: True Validator: TrueValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    protected $token;

    /**
     * @Assert\IsTrue(message = "El token es inválido")
     */
    public function isTokenValid()
    {
        return $this->token == $this->generateToken();
    }
}
```

IsFalse

Valida que un valor sea false, 0 o "0".

Aplicación: propiedades o métodos Opciones: message, payload Clase: IsFalse Validator: IsFalseValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\IsFalse(
```

```
     */
     public function isStateInvalid()
     {
        // ...
     }
}
```

Type

Valida si un valor es de un data type específico, por ejemplo, un array (por defecto es un string).

Aplicación: propiedades o métodos Opciones: message, payload, type Clase: Type Validator: TypeValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Type(
     *     type="integer",
     *     message="El valor {{ value }} no es del {{ type }} válido."
     * )
     */
    protected $age;
}
```

2. String Constraints

Email

Valida que un valor sea una dirección de email válida. El valor se pasa a string antes de ser validado.

Aplicación: propiedades o métodos Opciones: strict, message, checkMX, checkHost, payload Clase: Email Validator: EmailValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Email(
     *     message = "El email '{{ value }}' no es un email válido.",
     *     checkMX = true
     * )
     */
     protected $email;
}
```

Valida la longitud de un string entre valores mínimos y/o máximos.

Aplicación: propiedades o métodos Opciones: min, max, charset, minMessage, maxMessage, exactMessage, payload Clase: Length Validator: LengthValidator

```
// src/AppBundle/Entity/Participant.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Participant
{
    /**
     * @Assert\Length(
     *      min = 2,
     *      max = 50,
     *      minMessage = "Tu nombre debe tener al menos {{ limit }} caracteres",
     *      maxMessage = "Tu nombre no puede ser mayor a {{ limit }} caracteres"
     * )
     */
     protected $firstName;
}
```

Url

Valida que un valor es una URL válida.

Aplicación: propiedades o métodos Opciones: message, protocols, payload, checkDNS, dnsMessage Clase: Url Validator: UrlValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Url()
     */
     protected $bioUrl;
}
```

Regex

Valida que un valor coincida con una expresión regular.

Aplicación: propiedades o métodos Opciones: pattern, htmlPattern, match, message, payload Clase: Regex Validator: RegexValidator

El valor debe empezar con una palabra:

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;
```

```
class Author
{
    /**
     * @Assert\Regex("/^\w+/")
     */
    protected $description;
}
```

El valor no puede contener un integer (match false indica lo contrario):

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Regex(
     *     pattern="/\d/",
     *     match=false,
     *     message="Your name cannot contain a number"
     * )
     */
    protected $firstName;
}
```

Ip

Valida que un valor es una dirección IP válida. Por defecto valida IPv4, pero puedes especificar IPv6 con la opción version.

Aplicación: propiedades o métodos Opciones: version, message, payload Clase: Ip Validator: IpValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Ip
     */
     protected $ipAddress;
}
```

Uuid

Valida que un valor es un Universally unique identifier (UUID) válido, según el RFC 4122. Puede moderarse la restricción con la opción strict y especificar las versiones aceptadas con versions.

Aplicación: propiedades o métodos Opciones: message, strict, versions, payload Clase: Uuid Validator: UuidValidator

```
use Symfony\Component\Validator\Constraints as Assert;

class File
{
    /**
     * @Assert\Uuid
     */
     protected $identifier;
}
```

3. Number Constraints

Range

Valida que un número dado está entre un número mínimo y máximo.

Aplicación: propiedades o métodos Opciones: min, max, minMessage, maxMessage, invalidMessage, payload Clase: Range Validator: RangeValidator

Rango con números:

```
// src/AppBundle/Entity/Participant.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Participant
{
    /**
     * @Assert\Range(
     *      min = 120,
     *      max = 180,
     *      minMessage = "Debes ser al menos {{ limit }}cm de alto para entrar",
     *      maxMessage = "No puedes ser más alto que {{ limit }}cm para entrar"
     * )
     */
     protected $height;
}
```

Rango con fechas:

```
// src/AppBundle/Entity/Event.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Event
{
    /**
     * @Assert\Range(
     *      min = "first day of January",
     *      max = "first day of January next year"
     * )
     */
```

4. Comparison Constraints

EqualTo

Valida que un valor sea igual a otro, definido en las opciones.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: EqualTo Validator: EqualToValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\EqualTo(
     *     value = 20
     * )
     */
    protected $age;
}
```

NotEqualTo

Valida que un valor no es igual a otro, definido en las opciones.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: NotEqualTo Validator: NotEqualToValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\NotEqualTo(
     *     value = 15
     * )
     */
    protected $age;
}
```

IdenticalTo

Valida que un valor sea idéntico a otro, definido en las opciones (idéntico significa que se compara con triple igualador ===).

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: IdenticalTo Validator: IdenticalToValidator

```
use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\IdenticalTo(
     *     value = 20
     * )
     */
    protected $age;
}
```

NotIdenticalTo

Valida que un valor no sea idéntico a otro, definido en las opciones.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: NotIdenticalTo Validator: NotIdenticalToValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\NotIdenticalTo(
     *     value = 15
     * )
     */
    protected $age;
}
```

LessThan

Valida que una valor sea menor que otro, definido en las opciones. Soporta strings, números u objetos.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: LessThan Validator: LessThanValidator

Comparación de números:

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\LessThan(
     *     value = 80
     * )
     */
```

Comparación de fechas de objetos DateTime frente a un date string aceptado por el constructor de DateTime:

```
// src/AppBundle/Entity/Order.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Order
{
    /**
     * @Assert\GreaterThan("today")
     */
    protected $deliveryDate;
}
```

LessThanOrEqual

Valida que un valor sea igual o menor que otro, definido en las opciones. Soporta lo mismo que LessThan.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: LessThanOrEqual Validator: LessThanOrEqualValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\LessThanOrEqual(
     *     value = 80
     * )
     */
    protected $age;
}
```

GreaterThan

Valida que un valor es mayor que otro, definido en las opciones. También soporta strings, números u objetos.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: GreaterThan Validator: GreaterThanValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\GreaterThan(
```

```
     protected $age;
}
```

GreaterThanOrEqual

Valida que un valor es mayor o igual que otro, definido en las opciones. Soporta lo mismo que GreaterThan.

Aplicación: propiedades o métodos Opciones: value, message, payload Clase: GreaterThanOrEqual Validator: GreaterThanOrEqualValidator

```
// src/AppBundle/Entity/Person.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Person
{
    /**
     * @Assert\GreaterThanOrEqual(
     *     value = 18
     * )
     */
    protected $age;
}
```

5. Date Constraints

Date

Valida que un valor sea una fecha válida, ya sea un objeto DateTime o un string que sigue un formato YYYY-MM-DD válido.

Aplicación: propiedades o métodos Opciones: message, payload Clase: GreaterThanOrEqual Validator: GreaterThanOrEqualValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Date()
     */
     protected $birthday;
}
```

DateTime

Valida que un valor sea "datetime", ya sea un objeto DateTime o un string que sigue un formato YYYY-MM-DD HH:MM:SS válido.

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\DateTime()
     */
     protected $createdAt;
}
```

Time

Valida que un valor es una hora válida, ya sea un objeto DateTime o un string que sigue un formato "HH:MM:SS" válido.

Aplicación: propiedades o métodos Opciones: message, payload Clase: Time Validator: TimeValidator

```
// src/AppBundle/Entity/Event.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Event
{
    /**
     * @Assert\Time()
     */
     protected $startsAt;
}
```

6. Collection Constraints

Choice

Valida que un valor es uno de los valores proporcionados por el conjunto de choices. También puede validarse que cada uno de los valores de un array es una de esas choices.

Aplicación: propiedades o métodos Opciones: choices, callback, multiple, min, max, message, multipleMessage, minMessage, maxMessage, strict, payload Clase: Choice Validator: ChoiceValidator

Uso básico:

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
```

```
    protected $gender;
}
```

Podemos facilitar un callback que devuelve un array, en este caso se trata de getGenders de la clase Util:

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Choice(callback = {"Util", "getGenders"})
     */
    protected $gender;
}
```

Collection

Este constraint se emplea cuando los datos son una colección (un array u objeto que implementa Traversable y ArrayAccess), pero se quiere validar diferentes keys de la colección de formas distintas. Por ejemplo, si queremos validar el key email con un constraint Email y el key inventory con un constraint Range.

Aplicación: propiedades o métodos Opciones: fields, allowExtraFields, extraFieldsMessage, allowMissingFields, missingFieldsMessage, payload Clase: Collection Validator: CollectionValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Collection(
     *     fields = {
     *         "personal_email" = @Assert\Email,
     *         "short_bio" = {
     *             @Assert\NotBlank(),
     *             @Assert\Length(
     *                 max = 100,
     *                 maxMessage = "Tu bio es demasiado larga!"
     *             )
     *         }
     *     },
     *     allowMissingFields = true
     * )
     */
     protected $profileData = array(
         'personal_email',
         'short_bio',
     );
}
```

Valida que el número de elementos de una colección (un array o un objeto que implementa Countable) está entre un número mínimo y máximo.

Aplicación: propiedades o métodos Opciones: mix, max, minMessage, maxMessage, exactMessage, payload Clase: Count Validator: CountValidator

```
// src/AppBundle/Entity/Participant.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Participant
{
    /**
     * @Assert\Count(
     *      min = "1",
     *      max = "5",
     *      minMessage = "Debes especificar al menos un email",
     *      maxMessage = "No puedes especificar más de {{ limit }} emails"
     * )
     */
     protected $emails = array();
}
```

UniqueEntity

Valida que un campo (o campos) en una entidad Doctrine es (o son) única. Esto se suele emplear por ejemplo para evitar que un usuario se registre en el sistema con un email que ya existe.

Aplicación: clase Opciones: fields, message, em, repositoryMethod, errorPath, ignoreNull, payload Clase: UniqueEntity Validator: UniqueEntityValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;
use Doctrine\ORM\Mapping as ORM;

// DON'T forget this use statement!!!
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

/**
 * @ORM\Entity
 * @UniqueEntity("email")
 */
class Author
{
    /**
     * @var string $email
     *
     * @ORM\Column(name="email", type="string", length=255, unique=true)
     * @Assert\Email()
     */
    protected $email;

    // ...
}
```

Valida que un valor sea un idioma válido (Unicode Language Identifier).

Aplicación: propiedad o método Opciones: message, payload Clase: Language Validator: LanguageValidator

```
// src/AppBundle/Entity/User.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class User
{
    /**
     * @Assert\Language()
     */
     protected $preferredLanguage;
}
```

Locale

Valida que un valor sea un locale válido, ya sea con las dos letras del idioma (fr) o con cuatro especificando el país (fr_FR).

Aplicación: propiedad o método Opciones: message, payload Clase: Locale Validator: LocaleValidator

```
// src/AppBundle/Entity/User.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class User
{
    /**
     * @Assert\Locale()
     */
     protected $locale;
}
```

Country

Valida que un valor sea un códido válido de país según el estandard ISO 3166-1 alpha-2 ('ES', 'FR', 'PT', etc..).

Aplicación: propiedad o método Opciones: message, payload Clase: Country Validator: CountryValidator

```
// src/AppBundle/Entity/User.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class User
{
    /**
     * @Assert\Country()
     */
```

7. File Constraints

File

Valida que un valor sea un archivo válido, que puede ser un string path a un archivo existente o un objeto File válido (incluyendo los objetos de la clase UploadedFile). Es normalmente usado en formularios con el campo de formulario FileType.

Aplicación: propiedad o método Opciones: maxsize, binaryFormat, mimeTypes, maxSizeMessage, mimeTypesMessage, disallowEmptyMessage, notFoundMessage, notReadableMessage, uploadIniSizeErrorMessage, uploadFormSizeErrorMessage, uploadErrorMessage, payload Clase: File Validator: FileValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\File(
     *     maxSize = "1024k",
     *     mimeTypes = {"application/pdf", "application/x-pdf"},
     *     mimeTypesMessage = "Por favor sube un PDF válido"
     * )
     */
    protected $bioFile;
}
```

Image

Funciona como el file constraint, pero las opciones mimeTypes y mimeTypesMessage están ya configuradas para trabajar con archivos de imagen.

Aplicación: propiedad o método Opciones: mimeTypes, minWidth, maxWidth, maxHeight, minHeight, maxRatio, minRatio, allowSquare, allowLandscape, allowPortrait, mimeTypesMessage, sizeNotDetectedMessage, maxWidthMessage, minWidthMessage, maxHeightMessage, minHeightMessage, maxRatioMessage, minRatioMessage, allowSquareMessage, allowLandscapeMessage, allowPortraitMessage Clase: Image Validator: ImageValidator

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Image(
     *     minWidth = 200,
     *     maxWidth = 400,
     *     minHeight = 200,
     *     maxHeight = 400
```

```
    protected $headshot;
}
```

8. Financial and other Number Constraints

Bic

Valida si un valor tiene un formato BIC (Business Identifier Code) correcto (también llamado SWIFT). BIC es un acuerdo internacional que identifica instituciones financieras y no financieras con un valor único.

Aplicación: propiedad o método Opciones: message, payload Clase: Bic Validator: BicValidator

```
// src/AppBundle/Entity/Transaction.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Transaction
{
    /**
     * @Assert\Bic()
     */
    protected $businessIdentifierCode;
}
```

CardScheme

Valida que un número de tarjeta de crédito sea válido según la compañía de tarjetas de crédito.

Aplicación: propiedad o método Opciones: schemes, message, payload Clase: CardScheme Validator: CardSchemeValidator

```
// src/AppBundle/Entity/Transaction.php
namespace AppBundle\Entity\Transaction;

use Symfony\Component\Validator\Constraints as Assert;

class Transaction
{
    /**
     * @Assert\CardScheme(
     *     schemes={"VISA"},
     *     message="El número de tu tarjeta de crédito no es válido."
     * )
     */
    protected $cardNumber;
}
```

Currency

Valida que un valor sea un nombre de moneda válido según el estandard 3-letter ISO 4217 ('EUR', 'GBP', ...).

Aplicación: propiedad o método Opciones: message, payload Clase: Currency Validator: CurrencyValidator

```
use Symfony\Component\Validator\Constraints as Assert;

class Order
{
    /**
     * @Assert\Currency
     */
    protected $currency;
}
```

Luhn

Este constraint asegura que un número de tarjeta de crédito pasa el algoritmo Luhn.

Aplicación: propiedad o método Opciones: message, payload Clase: Luhn Validator: LuhnValidator

```
// src/AppBundle/Entity/Transaction.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Transaction
{
    /**
     * @Assert\Luhn(message = "Por favor comprueba el número de tu tarjeta de crédito.")
     */
    protected $cardNumber;
}
```

Iban

Este constraint comprueba que el número de cuenta de banco tiene el formato apropiado IBAN (International Bank Account Number). IBAN es un acuerdo internacional que identifica cuentas de banco en todo el mundo.

Aplicación: propiedad o método Opciones: message, payload Clase: Iban Validator: IbanValidator

```
// src/AppBundle/Entity/Transaction.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Transaction
{
    /**
     * @Assert\Iban(
     *     message="Este IBAN no es válido."
     * )
     */
    protected $bankAccountNumber;
}
```

Isbn

Aplicación: propiedad o método Opciones: type, message, isbn10Message, isbn13Message, bothIsbnMessage, payload Clase: Isbn Validator: IsbnValidator

```
// src/AppBundle/Entity/Book.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Book
{
    /**
     * @Assert\Isbn(
     *     type = isbn10,
     *     message: Este valor no es válido.
     * )
     */
    protected $isbn;
}
```

Issn

Este constraint valida que un valor sea un International Standar Serial Number (ISSN) correcto.

Aplicación: propiedad o método Opciones: message, caseSensitive, requireHyphen, payload Clase: Issn Validator: IssnValidator

```
// src/AppBundle/Entity/Journal.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Journal
{
    /**
     * @Assert\Issn
     */
     protected $issn;
}
```

9. Other Constraints

Callback

El objetivo del constraint callback es poder crear reglas de validación propias y asignar cualquier error de validación a campos específicos del objeto. Si empleas validación con formularios, puedes hacer que estos mensajes de error se muestren en cada campo.

Se especifica uno o más métodos callback, los cuales se llamarán durante el proceso de validación.

Un método callback no falla ni devuelve ningún valor. El callback añade validator violations.

Aplicación: class Opciones: callback, payload Clase: Callback Validator: CallbackValidator

Se añade el método validate con la validación callback en la clase:

```
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Validator\Context\ExecutionContextInterface;

/**
* @Assert\Callback
*/
class Author
{
    public function validate(ExecutionContextInterface $context)
    {
        // ...
    }
}
```

La función callback puede ser como sigue:

```
    private $firstName;

    public function validate(ExecutionContextInterface $context)
    {
        // tenemos un array de nombres falsos
        $fakeNames = array(/* ... */);

        // comprobamos si el nombre es falso
        if (in_array($this->getFirstName(), $fakeNames)) {
            $context->buildViolation('Este nombre es completamente falso!')
                ->atPath('firstName')
                ->addViolation();
        }
    }
```

También se pueden emplear callbacks estáticos y externos.

Expression

Este constraint te permite usar una expresión para una validación más compleja y dinámica.

Aplicación: clase, método o propiedad Opciones: expression, message, payload Clase: Expression Validator: ExpressionValidator

Imagina que tienes una clase BlogPost con las propiedades category y isTechnicalPost:

```
// src/AppBundle/Model/BlogPost.php
namespace AppBundle\Model;

use Symfony\Component\Validator\Constraints as Assert;

class BlogPost
{
    private $category;

    private $isTechnicalPost;

    // ...

    public function getCategory()
```

```
    }

    public function setIsTechnicalPost($isTechnicalPost)
    {
        $this->isTechnicalPost = $isTechnicalPost;
    }

    // ...
}
```

Para validar el objeto, tenemos ciertos requisitos específicos:

1. Si isTechnicalPost es true, category debe ser php o symfony.
2. Si isTechnicalPost es false, category puede ser cualquier cosa.

Una forma de hacer esto posible es con el constraint Expression:

```
// src/AppBundle/Model/BlogPost.php
namespace AppBundle\Model;

use Symfony\Component\Validator\Constraints as Assert;

/**
 * @Assert\Expression(
 *     "this.getCategory() in ['php', 'symfony'] or !this.isTechnicalPost()",
 *     message="If this is a tech post, the category should be either php or symfony!"
 * )
 */
class BlogPost
{
    // ...
}
```

Se utiliza la sintaxis de expresión del componente ExpressionLanguage.

All

Cuando se aplica a un array (o a un objeto Traversable), este constraint permite aplicar una colección de constraints a cada elemento del array.

Aplicación: método o propiedad Opciones: constraints, payload Clase: All Validator: AllValidator

```
// src/AppBundle/Entity/User.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class User
{
    /**
     * @Assert\All({
     *     @Assert\NotBlank,
     *     @Assert\Length(min = 5)
     * })
     */
     protected $favoriteColors = array();
}
```

Este constraint comprueba si el valor de input es igual al password del usuario actual. Es útil en un formulario donde un usuario puede cambiar su password, pero se requiere que inserte su antiguo password por seguridad.

No debe usarse como login, ya que se hace automáticamente desde el sistema de seguridad.

Aplicación: método o propiedad Opciones: message, payload Clase: UserPassword Validator: UserPasswordValidator

Ejemplo de una clase PasswordChange que se usa en un formulario donde el usuario puede cambiar su password insertando el antiguo y uno nuevo:

```
// src/AppBundle/Form/Model/ChangePassword.php
namespace AppBundle\Form\Model;

use Symfony\Component\Security\Core\Validator\Constraints as SecurityAssert;

class ChangePassword
{
    /**
     * @SecurityAssert\UserPassword(
     *     message = "Wrong value for your current password"
     * )
     */
     protected $oldPassword;
}
```

Valid

Este constraint se usa para la validación de objetos que están insertados como propiedades en un objeto. Permite validar un objeto y todos los subobjetos anidados.

Aplicación: método o propiedad Opciones: traverse, payload Clase: Valid

Por defecto la opción _errorbubblig está activada para el collection Field Type, que pasa los errores al formulario parent. Si quieres enlazar los errores a donde realmente ocurren establece a _errorbubbling como false.

Si tenemos dos clases, Author y Address, donde Author tiene una propiedad $address que es un objeto Address:

```
// src/AppBundle/Entity/Address.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Address
{
    /**
     * @Assert\NotBlank()
     */
    protected $street;

    /**
```

```
     */
    protected $zipCode;
}

// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\NotBlank
     * @Assert\Length(min = 4)
     */
    protected $firstName;

    /**
     * @Assert\NotBlank
     */
    protected $lastName;

    protected $address;
}
```

Con esta configuración, podemos validar un author con un address inválido. Para ello, añadimos el Valid constraint a la propiedad $address.

```
// src/AppBundle/Entity/Author.php
namespace AppBundle\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Valid
     */
    protected $address;
}
```

10. Configuración opcional

Algunos constraints, como NotBlank son muy simples, mientas que otros, como Choice, tienen varias opciones de configuración. Por ejemplo una clase Author tiene las choices male, female y other para la propiedad género:

```
// ...
use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Choice(
     *     choices = { "male", "female", "other" },
     *     message = "Choose a valid gender."
     * )
```

```
    // ...
}
```

Las opciones del constraint siempre se pueden pasar como un array. Pero además algunos constraints permiten pasar el valor de una opción por defecto en lugar del array. Por ejemplo en el constraint Choice se puede hacer:

```
// src/AppBundle/Entity/Author.php

// ...
use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\Choice({"male", "female", "other"})
     */
    protected $gender;

    // ...
}
```

Para asegurarte de cuáles permiten hacer esto puedes mirar la API.