Register FieldType

Introduction

This document explains how to register a custom FieldType in eZ Publish 5. It will not contain the development part as it is already covered in the API section.

Please be sure you first have read the basic documentation on how to develop a custom FieldType.

Service container configuration

> To be able to declare a FieldType, you need to have registered a bundle in your application kernel.
>
> This bundle needs to expose some configuration for the service container somehow (read related Symfony documentation)

Basic configuration

> This part relates to the base FieldType class that interacts with the Publish API.

Let's take a basic example from `ezstring` configuration.

```
parameters:
    ezpublish.fieldType.ezstring.class: eZ\Publish\Core\FieldType\TextLine\Type

services:
    ezpublish.fieldType.ezstring:
        class: %ezpublish.fieldType.ezstring.class%
        parent: ezpublish.fieldType
        tags:
            - {name: ezpublish.fieldType, alias: ezstring}
```

So far, this is a regular service configuration but 2 parts worth particular attention.

- `parent`

  As described in the Symfony Dependency Injection Component documentation, the `parent` config key indicates that you want your service to inherit from the parent's dependencies, including constructor arguments and method calls. This is actually a helper avoiding repetition in your field type configuration and keeping consistency between all field types.

- `tags`

  Tagging your field type service with **`ezpublish.fieldType`** is mandatory to be recognized by the API loader as a regular field type, the `alias` key being simply the *fieldTypeIdentifier* (formerly called *datatype string*)

> Basic field types configuration is located in EzPublishCoreBundle/Resources/config/fieldtypes.yml.

Legacy Storage Engine

Converter

As stated in Field Type API & best practices, a conversion of Field Type values is needed in order to properly store the data into the *old* database schema (aka *Legacy Storage*).

Those converters also need to be correctly exposed as services.

Field Type converter for ezstring

```
parameters:
    ezpublish.fieldType.ezstring.converter.class:
eZ\Publish\Core\Persistence\Legacy\Content\FieldValue\Converter\TextLine

services:
    ezpublish.fieldType.ezstring.converter:
        class: %ezpublish.fieldType.ezstring.converter.class%
        tags:
            - {name: ezpublish.storageEngine.legacy.converter, alias: ezstring, lazy:
true, callback: '::create'}
```

Here again we need to tag our converter service, with **`ezpublish.storageEngine.legacy.converter`** tag this time.

As for the tag attributes:

Attribute name	Usage
`alias`	Represents the *fieldTypeIdentifier* (just like for the FieldType service)
`lazy`	Boolean indicating if the converter should be lazy loaded or not. Performance wise, it is recommended to set it to **true** unless you have very specific reasons.
`callback`	If `lazy` is set to true, it represents the callback that will be called to build the converter. Any valid callback can be used. Note that if the callback is defined in the converter class, the class name can be omitted. This way, in the example above, the full callback will be resolved to `eZ\Publish\Core\Persistence\Legacy\Content\FieldValue\Converter\TextLine::create`

The converter configuration for basic field types are located in EzPublishCoreBundle/Resources/config/storage_engines.yml.

External storage

A FieldType has the ability to store its value (or part of it) in external data sources. This is made possible through the `eZ\Publish\SPI\FieldType\FieldStorage` interface. Thus, if one wants to use this functionality, he needs to define a service implementing this interface and tagged as **`ezpublish.fieldType.externalStorageHandler`** to be recognized by the Repository.

Here is an example for **ezurl** field type:

External storage handler for ezurl

```
parameters:
    ezpublish.fieldType.ezurl.externalStorage.class:
eZ\Publish\Core\FieldType\Url\UrlStorage

services:
    ezpublish.fieldType.ezurl.externalStorage:
        class: %ezpublish.fieldType.ezurl.externalStorage.class%
        tags:
            - {name: ezpublish.fieldType.externalStorageHandler, alias: ezurl}
```

The configuration is straight forward. Nothing specific except the **`ezpublish.fieldType.externalStorageHandler`** tag, the `alias` attribut e still begin the *fieldTypeIdentifier*.

> External storage configuration for basic field types is located in EzPublishCoreBundle/Resources/config/fieldtypes.yml.

Gateway based storage

As stated in the FieldType best practices, in order to be storage agnostic and external storage handler should use a *storage gateway*. This can be done by implementing another service implementing `eZ\Publish\Core\FieldType\StorageGateway` and being tagged as **`ezpublish.fieldType.externalStorageHandler.gateway`**.

Storage gateway for ezurl

```
parameters:
    ezpublish.fieldType.ezurl.storage_gateway.class:
eZ\Publish\Core\FieldType\Url\UrlStorage\Gateway\LegacyStorage

services:
    ezpublish.fieldType.ezurl.storage_gateway:
        class: %ezpublish.fieldType.ezurl.storage_gateway.class%
        tags:
            - {name: ezpublish.fieldType.externalStorageHandler.gateway, alias: ezurl,
identifier: LegacyStorage}
```

Attribute name	Usage
`alias`	Represents the *fieldTypeIdentifier* (just like for the FieldType service)
`identifier`	Identifier for the gateway. Must be unique per storage engine. *LegacyStorage* is the convention name for Legacy Storage Engine.

> For this to work properly, your storage handler must inherit from `eZ\Publish\Core\FieldType\GatewayBasedStorage`.
>
> Also note that there can be several gateways per field type (one per storage engine basically).
>
> The gateway configuration for basic field types are located in EzPublishCoreBundle/Resources/config/storage_engines.yml.