

# Implementing the Legacy Storage Engine Converter

So far, our type's value is represented by the `Tweet\Value` class. It holds a semantic representation of the type's data: an url, and author url, and the tweet's content.

The next step is to tell the system how to actually *store* this.

## About converters

Unlike eZ Publish Legacy, eZ Publish 5 supports (by design) multiple storage engines. The main, and almost only one right now is the Legacy Storage Engine, based on the legacy database, with a new implementation. Since each storage engine may have its own way of storing data, we need to map each `FieldType` value to something legacy can understand. To do this, we will implement a `FieldType Converter`, each storage engine defining its own interface for those.

## Legacy FieldType converters

The legacy storage engine uses the `ezcontentobject_attribute` table to store field values, and `ezcontentclass_attribute` to store field definition values (settings, etc). They're both based on the same principle. Each row represents a `Field` or a `FieldDefinition`, and offers several free fields, of different types, where the type can store its data:

- `ezcontentobject_attribute` offers three fields for this purpose: `data_int`, `data_text` and `data_float`
- `ezcontentclass_attribute` offers a few more: four `data_int` (`data_int1` to `data_int4`) fields, four `data_float` (`data_float1` to `data_float5`) ones, and five `data_text` (`data_text1` to `data_text5`).

Each type is free to use those fields in any way it requires. Converters will map a field's semantic values to the fields described above, for both settings (validation + configuration) as well as value.

## Implementing Tweet\LegacyConverter

The Converter will be placed along with the Type and Value definitions (the Kernel stores them inside the Legacy Storage Engine structure): `eZ/Publish/FieldType/Tweet/LegacyConverter.php`. A Legacy Converter must implement the `eZ\Publish\Core\Persistence\Legacy\Content\FieldValue\Converter` interface:

```
namespace EzSystems\TweetFieldTypeBundle\eZ\Publish\FieldType\Tweet;

use eZ\Publish\Core\Persistence\Legacy\Content\FieldValue\Converter;

class LegacyConverter implements Converter
{
}
```

The Converter interface expects us to implement five methods:

- `toStorageValue()` & `toFieldValue()`  
used to convert an API field value to a legacy storage value, and a legacy storage value to an API field value.
- `toStorageFieldDefinition()` & `toFieldDefinition()`  
used to convert a field definition to a legacy one, and a stored legacy field definition to an API field definition
- `getIndexColumn()`  
Tell the API which legacy DB field should be used to sort & filter content, either `sort_key_string` or `sort_key_int`

## Implementing Field Value converters: toFieldValue() and toStorageValue()

As said above, those two methods are used to convert from a `Field` to a value Legacy can store, and the other way around.

We have defined that we wanted to store the tweet's URL in `data_text`, and that sorting would be done on the `username-status-tweetid` string we extract in `getName()` and `getSortInfo()`.

`toStorageValue()` will fill the provided `eZ\Publish\Core\Persistence\Legacy\Content\StorageFieldValue` from a `Tweet\Value`, while `toFieldValue()` will do the exact opposite:

```

use eZ\Publish\Core\Persistence\Legacy\Content\StorageFieldValue;
use eZ\Publish\SPI\Persistence\Content\FieldValue;

// [...]

public function toStorageValue( FieldValue $value, StorageFieldValue
$storageFieldValue )
{
    $storageFieldValue->dataText = $value->data;
    $storageFieldValue->sortKeyString = $value->sortKey;
}
public function toFieldValue( StorageFieldValue $value, FieldValue $fieldValue )
{
    $fieldValue->data = $value->dataText;
    $fieldValue->sortKey = $value->sortKeyString;
}

```

With these two methods, the legacy storage engine is able to convert a `Tweet\Value` into legacy data, and legacy data back into a `Tweet\Value` object.

## Implementing Field Definition converters: `toStorageFieldDefinition()` and `toFieldDefinition()`

The first two methods we have implemented apply to a Field's value. But we also need to convert our Field's definition. For example, a `TextLine`'s max length, or any `FieldDefinition` option.

This is done using `toStorageDefinition()`, that converts a `FieldDefinition` into a `StorageFieldDefinition`. `toFieldDefinition()` does the opposite. In our case, we actually don't need to implement those methods since our `Tweet` Type doesn't have settings:

```

use eZ\Publish\Core\Persistence\Legacy\Content\StorageFieldDefinition;
use eZ\Publish\SPI\Persistence\Content\Type\FieldDefinition;

// [...]

public function toStorageFieldDefinition( FieldDefinition $fieldDef,
StorageFieldDefinition $storageDef )
{
}

public function toFieldDefinition( StorageFieldDefinition $storageDef, FieldDefinition
$fieldDef )
{
}

```

## Implementing `getIndexColumn()`

In `toFieldValue()` and `toStorageValue()`, we have used the `sortKeyString` property from `StorageFieldValue`. `getIndexColumn()` will tell provide the legacy storage engine the type of index / sort column it should use: `string` (`sort_key_string`) or `int` (`sort_key_int`). Depending on which one is returned, the system will either use the `sortKeyString` or the `sortKeyInt` properties from the `StorageFieldValue`.

```
public function getIndexColumn()
{
    return 'sort_key_string';
}
```

## Registering the converter

Just like a Type, a Legacy Converter needs to be registered and tagged in the service container.

The tag is `ezpublish.storageEngine.legacy.converter`, and it requires an `alias` attribute to be set to the FieldType identifier (`eztweet`). Let's add this block to `Resources/config/services.yml`:

### Resources/config/services.yml

```
services:
    ezsystems.tweetbundle.fieldType.eztweet.converter:
        class:
            EzSystems\TweetFieldTypeBundle\ez\Publish\FieldType\Tweet\LegacyConverter
        tags:
            - {name: ezpublish.storageEngine.legacy.converter, alias: eztweet}
```

---

Previous: [Registering the FieldType as a service](#)

Next: [Template implementation](#)