3. Managing Content

In the following recipes, you will see how to create Content, including complex fields like XmlText or Image.

Identifying to the repository with a login and a password

As seen earlier, the Repository executes operations with a user's credentials. In a web context, the currently logged in user is automatically identified. In a command line context, you need to manually log a user in. We have already seen how to manually load and set a user using its ID. If you would like to identify a user using his username and password instead, this can be achieved as follows.

authentication

```
$user = $userService->loadUserByCredentials( $user, $password );
$repository->setCurrentUser( $user );
```

Creating content

Full code
https://github.com/ezsystems/CookbookBundle/blob/master/Command/CreateContentCommand.php

We will now see how to create Content using the Public API. This example will work with the default Folder (ID 1) Content Type from eZ Publish.

```
/** @var $repository \eZ\Publish\API\Repository\Repository */
$repository = $this->getContainer()->get( 'ezpublish.api.repository' );
$contentService = $repository->getContentService();
$locationService = $repository->getLocationService();
$contentTypeService = $repository->getContentTypeService();
```

We first need the required services. In this case: `ContentService`, `LocationService` and `ContentTypeService`.

The ContentCreateStruct

As explained in the Public API Basics, Value Objects are read only. Dedicated objects are provided for Update and Create operations: structs, like ContentCreateStruct or UpdateCreateStruct. In this case, we need to use a ContentCreateStruct.

```
$contentType = $contentTypeService->loadContentTypeByIdentifier( 'article' );
$contentCreateStruct = $contentService->newContentCreateStruct( $contentType, 'eng-GB'
);
```

We first need to get the `ContentType` we want to create a `Content` with. To do so, we use `ContentTypeService::loadContentTypeByIdentifier()`, with the wanted `ContentType` identifier, like 'article'. We finally get a ContentTypeCreateStruct using `ContentService::newC`

ontentCreateStruct(), providing the ContentType and a Locale Code (eng-GB).

Setting the fields values

```
$contentCreateStruct->setField( 'title', 'My title' );
$contentCreateStruct->setField( 'intro', $intro );
$contentCreateStruct->setField( 'body', $body );
```

Using our create struct, we can now set the values for our Content's fields, using the `setField()` method. For now, we will just set the title. `setField()` for a TextLine Field simply expects a string as input argument. More complex FieldTypes, like Author or Image, expect different input values.

> The `ContentCreateStruct::setField()` method can take several type of arguments.
>
> In any case, whatever the FieldType is, a Value of this type can be provided. For instance, a TextLine\Value can be provided for a TextLine\Type. Depending on the FieldType implementation itself, more specifically on the fromHash() method every FieldType implements, various arrays can be accepted, as well as primitive types, depending on the Type.

Setting the Location

In order to set a Location for our object, we must instantiate a `LocationCreateStruct`. This is done with LocationService::newLocationCreateStruct(), with the new Location's parent ID as an argument.

```
$locationCreateStruct = $locationService->newLocationCreateStruct( 2 );
```

Creating and publishing

To actually create our Content in the Repository, we need to use ContentService::createContent(). This method expects a `ContentCreateStruct`, as well as a `LocationCreateStruct`. We have created both in the previous steps.

```
$draft = $contentService->createContent( $contentCreateStruct, array(
$locationCreateStruct ) );
$content = $contentService->publishVersion( $draft->versionInfo );
```

The LocationCreateStruct is provided as an array, since a Content can have multiple locations.

`createContent()` returns a new Content Value Object, with one version that has the DRAFT status. To make this Content visible, we need to publish it. This is done using `ContentService::publishVersion()`. This method expects a `VersionInfo` object as its parameter. In our case, we simply use the current version from `$draft`, with the `versionInfo` property.

Updating Content

> **Full code**
> https://github.com/ezsystems/CookbookBundle/blob/master/Command/UpdateContentCommand.php

We will now see how the previously created Content can be updated. To do so, we will create a new draft for our Content, update it using a `ContentUpdateStruct`, and publish the updated Version.

```
$contentInfo = $contentService->loadContentInfo( $contentId );
$contentDraft = $contentService->createContentDraft( $contentInfo );
```

To create our draft, we need to load the Content's ContentInfo using `ContentService::loadContentInfo()`. We can then use `ContentService::createContentDraft()` to add a new Draft to our Content.

```
// instantiate a content update struct and set the new fields
$contentUpdateStruct = $contentService->newContentUpdateStruct();
$contentUpdateStruct->initialLanguageCode = 'eng-GB'; // set language for new version
$contentUpdateStruct->setField( 'title', $newTitle );
$contentUpdateStruct->setField( 'body', $newBody );
```

To set the new values for this version, we request a `ContentUpdateStruct` from the `ContentService` using the `newContentUpdateStruct()` method. Updating the values hasn't changed: we use the `setField()` method.

```
$contentDraft = $contentService->updateContent( $contentDraft->versionInfo,
$contentUpdateStruct );
$content = $contentService->publishVersion( $contentDraft->versionInfo );
```

We can now use `ContentService::updateContent()` to apply our `ContentUpdateStruct` to our draft's `VersionInfo`. Publishing is done exactly the same way as for a new content, using `ContentService::publishVersion()`.

Handling translations

In the two previous examples, you have seen that we set the ContentUpdateStruct's initialLanguageCode property. To translate an object to a new language, set the locale to a new one.

translating

```
$contentUpdateStruct->initialLanguageCode = 'ger-DE';
$contentUpdateStruct->setField( 'title', $newtitle );
$contentUpdateStruct->setField( 'body', $newbody );
```

It is possible to create or update content in multiple languages at once. There is one restriction: only one language can be set a version's language. This language is the one that will get a flag in the back office. However, you can set values in other languages for your attributes, using the setField method's third argument.

update multiple languages

```
// set one language for new version
$contentUpdateStruct->initialLanguageCode = 'fre-FR';

$contentUpdateStruct->setField( 'title', $newgermantitle, 'ger-DE' );
$contentUpdateStruct->setField( 'body', $newgermanbody, 'ger-DE' );

$contentUpdateStruct->setField( 'title', $newfrenchtitle );
$contentUpdateStruct->setField( 'body', $newfrenchbody );
```

Since we don't specify a locale for the last two fields, they are set for the `UpdateStruct`'s `initialLanguageCode`, fre-FR.

Creating Content containing an image

Full code
https://github.com/ezsystems/CookbookBundle/blob/master/Command/CreateImageCommand.php

As explained above, the setField() method can accept various values: an instance of the FieldType's Value class, a primitive type, or a hash. The last two depend on what the `Type::acceptValue()` method is build up to handle. TextLine can, for instance, accept a simple string as an input value. In this example, you will see how to set an Image value.

We assume that we use the default image class. Creating our Content, using the ContentType and a ContentCreateStruct, has been covered above, and can be found in the full code. Let's focus on how the image is provided.

```
$file = '/path/to/image.png';

$value = new \eZ\Publish\Core\FieldType\Image\Value(
    array(
        'path' => '/path/to/image.png',
        'fileSize' => filesize( '/path/to/image.png' ),
        'fileName' => basename( 'image.png' ),
        'alternativeText' => 'My image'
    )
);
$contentCreateStruct->setField( 'image', $value );
```

This time, we create our image by directly providing an `Image\Value` object. The values are directly provided to the constructor using a hash with predetermined keys that depend on each Type. In this case: the path where the image can be found, its size, the file name, and an alternative text.

Images also implement a static `fromString()` method that will, given a path to an image, return an `Image\Value` object.

```
$value = \eZ\Publish\Core\FieldType\Image\Value::fromString( '/path/to/image.png' );
```

But as said before, whatever you provide `setField()` with is sent to the `acceptValue()` method. This method really is the entry point to the input formats a FieldType accepts. In this case, you could have provided setField with either a hash, similar to the one we provided the Image\Value constructor with, or the path to your image, as a string.

```
$contentCreateStruct->setField( 'image', '/path/to/image.png' );

// or

$contentCreateStruct->setField( 'image', array(
    'path' => '/path/to/image.png',
    'fileSize' => filesize( '/path/to/image.png' ),
    'fileName' => basename( 'image.png' ),
    'alternativeText' => 'My image'
);
```

Create Content with XML Text

> **Full code**
> https://github.com/ezsystems/CookbookBundle/blob/master/Command/CreateXMLContentCommand.php

Another very commonly used FieldType is the rich text one, `XmlText`.

working with xml text

```
$xmlText = <<< EOX
<?xml version='1.0' encoding='utf-8'?>
<section>
<paragraph>This is a <strong>image test</strong></paragraph>
<paragraph><embed view='embed' size='medium' object_id='$imageId'/></paragraph>
</section>
EOX;
$contentCreateStruct->setField( 'body', $xmlText );
```

As for the last example above, we use the multiple formats accepted by `setField()`, and provide our XML string as is. The only accepted format as of 5.0 is internal XML, the one stored in the Legacy database.

> The XSD for the internal XML representation can be found in the kernel: https://github.com/ezsystems/ezpublish-kernel/blob/master/eZ/Publish/Core/FieldType/XmlText/Input/Resources/schemas/ezxml.xsd.

We embed an image in our XML, using the <embed> tag, providing an image Content ID as the `object_id` attribute.

> **Using a custom format as input**
> More input formats will be added later. The API for that is actually already available: you simply need to implement the `XmlText\Input` interface. It contains one method, `getInternalRepresentation()`, that must return an internal XML string. Create your own bundle, add your implementation to it, and use it in your code!
>
> ```
> $input = new \My\XmlText\CustomInput( 'My custom format string' );
> $contentCreateStruct->setField( 'body', $input );
> ```

Deleting Content

```
$contentService->deleteContent( $contentInfo );
```

`ContentService::deleteContent()` method expects a `ContentInfo` as an argument. It will delete the given Content, all of its Locations, as well as all of the Content's Locations' descendants and their associated Content. **Use with caution !**