5. Other recipes

Assigning section to content

> **Full code**
> https://github.com/ezsystems/CookbookBundle/tree/master/Command/AssignContentToSectionCommand.php

The Section a Content belongs to can be set during creation, using the `ContentCreateStruct::$sectionId` property. However, as for many Repository objects properties, the section can't be changed using a `ContentUpdateStruct`. The reason is still the same: changing a Content's section will affect the subtrees referenced by its Locations. For this reason, it is required that you use the SectionService to change the Section of a Content.

assign section to content

```
$contentInfo = $contentService->loadContentInfo( $contentId );
$section = $sectionService->loadSection( $sectionId );
$sectionService->assignSection( $contentInfo, $section );
```

This operation involves the `SectionService`, as well as the `ContentService`.

assign section to content

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

We use `ContentService::loadContentInfo()` to get the Content we want to update the Section for.

assign section to content

```
$section = $sectionService->loadSection( $sectionId );
```

SectionService::loadSection() is then used to load the Section we want to assign our Content to. Note that there is no SectionInfo object. Sections are quite simple, and we don't need to separate their metadata from their actual data. However, `SectionCreateStruct` and `SectionUpdateStruct` objects must still be used to create and update sections.

assign section to content

```
$sectionService->assignSection( $contentInfo, $section );
```

The actual update operation is done using SectionService::assignSection(), with the ContentInfo and the Section as arguments.

> SectionService::assignSection() won't return the updated Content, as it has no knowledge of those objects. To get the Content with the newly assigned Location, you need to reload the ContentInfo using `ContentService::loadContentInfo()`. This is also valid for descendants of Content. If you have any stored in your execution state, you need to reload them. Otherwise you would be using outdated Content data.

Creating a content type

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

Creating a `ContentType` is actually almost more complex than creating Content. It really isn't as common, and didn't "deserve" the same kind of API as Content did.

Let's split the code in three major parts.

```
try
{
    $contentTypeGroup = $contentTypeService->loadContentTypeGroupByIdentifier(
'content' );
}
catch ( \eZ\Publish\API\Repository\Exceptions\NotFoundException $e )
{
    $output->writeln( "content type group with identifier $groupIdentifier not found"
);
    return;
}


$contentTypeCreateStruct = $contentTypeService->newContentTypeCreateStruct(
'mycontenttype' );
$contentTypeCreateStruct->mainLanguageCode = 'eng-GB';
$contentTypeCreateStruct->nameSchema = '<title>';
$contentTypeCreateStruct->names = array(
    'eng-GB' => 'My content type'
);
$contentTypeCreateStruct->descriptions = array(
    'eng-GB' => 'Description for my content type',
);
```

First, we need to load the `ContentTypeGroup` our `ContentType` will be created in. We do this using `ContentTypeService::loadContentTypeGroupByIdentifier()`, which gives us back a `ContentTypeGroup` object. As for content, we then request a `ContentTypeCreateStruct` from the `ContentTypeService`, using `ContentTypeService::newContentTypeCreateStruct()`, with the desired identifier as the argument.

Using the create struct's properties, we can set the type's properties:

- the main language (`mainLanguageCode`) for the type is set to eng-GB,
- the content name generation pattern (`nameSchema`) is set to '<title>': content of this type will be named as their 'title' field.
- the human readable name for our type is set using the `names` property. We give it a hash, indexed by the locale ('eng-GB') the name is set in. This locale must exist in the system.
- the same way that we have set the `names` property, we can set human readable descriptions, again as hashes indexed by locale code.

The next big part is to add FieldDefinition objects to our ContentType.

```
// add a TextLine Field with identifier 'title'
$titleFieldCreateStruct = $contentTypeService->newFieldDefinitionCreateStruct(
'title', 'ezstring' );
$titleFieldCreateStruct->names = array( 'eng-GB' => 'Title' );
$titleFieldCreateStruct->descriptions = array( 'eng-GB' => 'The Title' );
$titleFieldCreateStruct->fieldGroup = 'content';
$titleFieldCreateStruct->position = 10;
$titleFieldCreateStruct->isTranslatable = true;
$titleFieldCreateStruct->isRequired = true;
$titleFieldCreateStruct->isSearchable = true;
$contentTypeCreateStruct->addFieldDefinition( $titleFieldCreateStruct );


// add a TextLine Field body field
$bodyFieldCreateStruct = $contentTypeService->newFieldDefinitionCreateStruct( 'body',
'ezstring' );
$bodyFieldCreateStruct->names = array( 'eng-GB' => 'Body' );
$bodyFieldCreateStruct->descriptions = array( 'eng-GB' => 'Description for Body' );
$bodyFieldCreateStruct->fieldGroup = 'content';
$bodyFieldCreateStruct->position = 20;
$bodyFieldCreateStruct->isTranslatable = true;
$bodyFieldCreateStruct->isRequired = true;
$bodyFieldCreateStruct->isSearchable = true;
$contentTypeCreateStruct->addFieldDefinition( $bodyFieldCreateStruct );
```

We need to create a `FieldDefinitionCreateStruct` object for each `FieldDefinition` our `ContentType` will be made of. Those objects are obtained using `ContentTypeService::newFieldDefinitionCreateStruct()`. This method expects the FieldDefinition identifier and its type as arguments. The identifiers match the ones from eZ Publish 4 (`ezstring` for TextLine, etc).

Each field's properties are set using the create struct's properties:

- `names` and `descriptions` are set using hashes indexed by the locale code, and with the name or description as an argument.
- The `fieldGroup` is set to 'content'
- Fields are ordered using the `position` property, ordered numerically in ascending order. We set it to an integer.
- The translatable, required ans searchable boolean flags are set using their respective property: `isTranslatable`, `isRequired` and `isSearchable`.

Once the properties for each create struct are set, the field is added to the ContentType create struct using `ContentTypeCreateStruct::addFieldDefinition()`.

```
try
{
    $contentTypeDraft = $contentTypeService->createContentType(
$contentTypeCreateStruct, array( $contentTypeGroup ) );
    $contentTypeService->publishContentTypeDraft( $contentTypeDraft );
}
catch ( \eZ\Publish\API\Repository\Exceptions\UnauthorizedException $e )
{
    $output->writeln( "<error>" . $e->getMessage() . "</error>" );
}
catch ( \eZ\Publish\API\Repository\Exceptions\ForbiddenException $e )
{
    $output->writeln( "<error>" . $e->getMessage() . "</error>" );
}
```

The last step is the same as for Content: we create a content type draft using `ContentTypeService::createContentType()`, with the `ContentTypeCreateStruct` and an array of `ContentTypeGroup` objects are arguments. We then publish the ContentType draft using `ContentTypeService::publishContentTypeDraft()`.