eZ Platform Public PHP API

The PHP API is also commonly referred to as the "*Public API*". Currently it exposes a Repository which allows you to create, read, update, manage and delete all objects available in eZ Platform, first and foremost content, but also related objects like Sections, Locations, Content Types, Content Type groups, languages and so on.

In this topic:

eZ Platform API Repository

This entity is the entry point to everything you will do with the Public API.

It will allow you to create, retrieve, update and delete all the eZ Platform objects, as well as Content Types, Sections, Content states. It is always obtained through the service container.

Obtaining the eZ Platform Repository via the service container

```
/** @var $repository
\eZ\Publish\API\Repository\Repository
$repository = $container->get(
'ezpublish.api.repository' );
```

By itself, the repository doesn't do much. It allows three types of operations: user authentication (getting / changing the current user), issuing transactions, and obtaining services.

Inline objects documentation
Pay attention to the inline phpdoc block in this code stub. It tells your IDE that `$repository` is an instance of `\eZ\Publish\API\Repository\Repository` . If your IDE supports this feature, you will get code completion on the `$repository` object. This helper is a huge time saver when it comes to learning about the eZ Platform API.

The service container

The above code snippet implies that the service container is available in the context you are writing your code in.

In controllers, this generally is done by extending the Symfony `Controller` class. It comes with a `get()` method that calls the service container. In command line scripts, it requires that you extend the `ContainerAwareCommand` base class instead of `Controller`. This class provides you with a `getContainer()` method that returns the service container.

Getting the repository from eZ Platform controllers
In order to make it even easier to obtain the repository from controllers code, eZ Platform controllers extend a custom Controller class that provides a `getRepository()` method which directly returns the repository from the service container.

You can and should of course do the same in your custom controllers.

Authentication

One of the responsibilities of the Repository is user authentication. Every action will be executed *as* a user. In the context of a normal eZ Platform execution, the logged in user will of course be the current one, identified via one of the available authentication methods. This user's permissions will affect the behavior of the Repository. The user may for example not be allowed to create Content, or view a particular Section.

Logging in to the Repository is covered in other recipes of the Cookbook.

Services

The main entry point to the repository's features are services. The Public API breaks down access to Content, User, Content Types and other features into various services. Those services are obtained via the Repository, using `get`[ServiceName]`()` methods: `getContentService()` , `getUserService()` , etc.

Throughout the Cookbook, you will be guided through the various capabilities those services have, and how you can use them to implement your projects.

Value objects

While Services provide interaction with the repository, the elements (Content, Users) they provide interaction with are provided as read-only Value Objects in the `eZ\Publish\Core\Repository\Values` namespace. Those objects are broken down into sub-namespaces: `Content`, `ContentType`, `User` and `ObjectState`, each sub-namespace containing a set of value objects, such as `Content\Content` or `User\Role` .

These objects are read-only by design. They are only meant to be used in order to fetch data from the repository. They come with their own properties, such as `$content->id`, `$location->hidden`, but also with methods that provide access to more related information, such as `Relation::getSourceContentInfo()` or `Role::getPolicies()`. By design, a value object will only give you access to data that is very closely related to it. More complex retrieval operations will require you to use the appropriate Service, using information from your Value Object.

Value info objects

Some complex Value Objects have an `Info` counterpart, like `ContentInfo`, the counterpart for `Content`. These objects are specific and provide you with lower-level information. For instance, `ContentInfo` will provide you with `currentVersionNo` or `remoteId`, while `Content` will let you retrieve Fields, the Content Type, or previous Versions.

They are provided by the API, but are **read only**, can't be modified and sent back. Creation and modification of Repository values is done using Create structs and Update structs.

Create and update structs

In order to update or create elements in the Repository, you will use structs. They are usually provided by the Service that manages the Value Objects you want to alter or create. For instance, the Content service has a `getContentCreateStruct()` method that returns a new `ContentCreateStruct` object. Equivalent methods exist for UpdateStruct objects as well, and for most Value Objects.

Using them is also covered in the Cookbook.