

Custom policies

PLATFORM >= 2015.09

Description

eZ content repository uses the concept of Roles and Policies in order to authorize a user to do something (e.g. read content).

- A Role is composed of Policies and can be assigned to a User or a User Group.
- A Policy is composed of a combination of **module** and **function** (e.g. `content/read`, `content` being the module and `read` being the function).
- Depending on **module** and **function** combination, a Policy can also be composed of Limitations.

It is possible for any bundle to expose available Policies via a `PolicyProvider` which can be added to `EzPublishCoreBundle`'s DIC extension.

PolicyProvider

A `PolicyProvider` is an object providing a hash containing declared modules, functions and limitations.

- Each Policy provider provides a collection of permission *modules*.
- Each module can provide *functions* (e.g. "content/read": "content" is the module, "read" is the function)
- Each function can provide a collection of Limitations.

Policies configuration hash contains declared these modules, functions and Limitations.

First level key is the module name, value is a hash of available functions, with function name as key.

Function value is an array of available Limitations, identified by the alias declared in `LimitationType` service tag.

If no Limitation is provided, value can be `null` or an empty array.

```
[
  "content" => [
    "read" => ["Class", "ParentClass", "Node", "Language"],
    "edit" => ["Class", "ParentClass", "Language"]
  ],
  "custom_module" => [
    "custom_function_1" => null,
    "custom_function_2" => ["CustomLimitation"]
  ],
]
```

Limitations need to be implemented as *limitation types* and declared as services identified with `ezpublish.limitationType` tag. Name provided in the hash for each Limitation is the same value set in `alias` attribute in the service tag.

Example

```

namespace Acme\FooBundle\AcmeFooBundle\Security;

use
eZ\Bundle\EzPublishCoreBundle\DependencyInjection\Configuration\ConfigBuilderInterface
;
use
eZ\Bundle\EzPublishCoreBundle\DependencyInjection\Security\PolicyProvider\PolicyProvid
erInterface;

class MyPolicyProvider implements PolicyProviderInterface
{
    public function addPolicies(ConfigBuilderInterface $configBuilder)
    {
        $configBuilder->addConfig([
            "custom_module" => [
                "custom_function_1" => null,
                "custom_function_2" => ["CustomLimitation"],
            ],
        ]);
    }
}

```

YamlPolicyProvider

An abstract class based on YAML is provided: `eZ\Bundle\EzPublishCoreBundle\DependencyInjection\Security\PolicyProvider\YamlPolicyProvider`.

It defines an abstract `getFiles()` method.

Extend `YamlPolicyProvider` and implement `getFiles()` to return absolute paths to your YAML files.

```

namespace Acme\FooBundle\AcmeFooBundle\Security;

use
eZ\Bundle\EzPublishCoreBundle\DependencyInjection\Security\PolicyProvider\YamlPolicyPr
ovider;

class MyPolicyProvider extends YamlPolicyProvider
{
    protected function getFiles()
    {
        return [
            __DIR__ . '/../Resources/config/policies.yml',
        ];
    }
}

```

AcmeFooBundle/Resources/config/policies.yml

```

custom_module:
    custom_function_1: ~
    custom_function_2: [CustomLimitation]

```

Extending existing policies

A `PolicyProvider` may provide new functions to a module, and additional `Limitations` to an existing function.

It is however strongly encouraged to add functions to your own `Policy` modules.

It is not possible to remove an existing module, function or limitation from a `Policy`.

Integrating the `PolicyProvider` into `EzPublishCoreBundle`

For a `PolicyProvider` to be active, it must be properly declared in `EzPublishCoreBundle`.

A bundle just has to retrieve `CoreBundle`'s DIC extension and call `addPolicyProvider()`. This must be done in the bundle's `build()` method.

```
namespace Acme\FooBundle\AcmeFooBundle;

use Symfony\Component\HttpKernel\Bundle\Bundle;

class AcmeFooBundle extends Bundle
{
    public function build(ContainerBuilder $container)
    {
        parent::build($container);

        // ...

        // Retrieve "ezpublish" container extension.
        $eZExtension = $container->getExtension('ezpublish');
        // Add the policy provider.
        $eZExtension->addPolicyProvider(new MyPolicyProvider());
    }
}
```

Core policies

Policies used internally in repository services are defined in `EzPublishCoreBundle/Resources/config/policies.yml`.

Related topics:

- [Permissions](#)
- [Limitations reference](#)