

Configuration

- The Basics
 - Example
- Dynamic configuration with the ConfigResolver
 - Scope
 - ConfigResolver Usage
 - Inject the ConfigResolver in your services
 - Custom locale configuration (feature available since 5.1)

The Basics

Important

Configuration is tightly related to the service container.

To fully understand the following content, you need to be aware of [Symfony's service container and its configuration](#).

Basic configuration handling in eZ Publish is similar to what is commonly possible with Symfony. Regarding this, you can define key/value pairs in your configuration files, under the main **parameters** key (like in **parameters.yml**).

Internally and by convention, keys follow a **dot syntax** where the different segments follow your configuration hierarchy. Keys are usually prefixed by a *namespace* corresponding to your application.

Values can be anything, **including arrays and deep hashes**.

eZ Publish core configuration is prefixed by **ezsettings** namespace, while *internal* configuration (not to be used directly) is prefixed by **epublish** namespace.

Example

Configuration

```
parameters:
  myapp.parameter.name: someValue
  myapp.boolean.param: true
  myapp.some.hash:
    foo: bar
    an_array: [apple, banana, pear]
```

Usage from a controller

```
// Inside a controller
$myParameter = $this->container->getParameter( 'myapp.parameter.name' );
```

Dynamic configuration with the ConfigResolver

In eZ Publish, it is fairly common to have different settings depending on the current siteaccess (e.g. languages, [view provider](#) configuration).

Scope

Dynamic configuration can be resolved depending on a *scope*.

Available scopes are (in order of precedence) :

1. *global*
2. *SiteAccess*
3. *SiteAccess group*
4. *default*

It gives the opportunity to define settings for a given siteaccess, for instance, like in the [legacy INI override system](#).

This mechanism is not limited to eZ Publish internal settings (aka **ezsettings namespace**) and is applicable for specific needs (bundle related, project related, etc).

ConfigResolver Usage

Dynamic configuration is handled by a **config resolver**. It consists in a service object mainly exposing `hasParameter()` and `getParameter()` methods. The idea is to check the different *scopes* available for a given *namespace* to find the appropriate parameter.

In order to work with the config resolver, your dynamic settings must comply internally to the following name format : `<namespace>.<scope>.parameter.name`.

Namespace + scope example

```
parameters:
  # Some internal configuration
  ezsettings.default.content.default_ttl: 60
  ezsettings.ezdemo_site.content.default_ttl: 3600

  # Here "myapp" is the namespace, followed by the siteaccess name as the parameter
  scope
  # Parameter "foo" will have a different value in ezdemo_site and ezdemo_site_admin
  myapp.ezdemo_site.foo: bar
  myapp.ezdemo_site_admin.foo: another value
  # Defining a default value, for other siteaccesses
  myapp.default.foo: Default value

  # Defining a global setting, available for all siteaccesses
  #myapp.global.some.setting: This is a global value
```

```

// Inside a controller, assuming siteaccess being "ezdemo_site"
/** @var $configResolver \eZ\Publish\Core\MVC\ConfigResolverInterface */
$configResolver = $this->getConfigResolver();

// ezsettings is the default namespace, so no need to precise it
// The following will resolve ezsettings.<siteaccessName>.content.default_ttl
// In the case of ezdemo_site, will return 3600.
// Otherwise it will return the value for ezsettings.default.content.default_ttl (60)
$locationViewSetting = $configResolver->getParameter( 'content.default_ttl' );

$fooSetting = $configResolver->getParameter( 'foo', 'myapp' );
// $fooSetting's value will be 'bar'

// Force scope
$fooSettingAdmin = $configResolver->getParameter( 'foo', 'myapp', 'ezdemo_site_admin' );
// $fooSetting's value will be 'another value'

// Note that the same applies for hasParameter()

```

Both `getParameter()` and `hasParameter()` can take 3 different arguments:

1. `$paramName` (i.e. the name of the parameter you need)
2. `$namespace` (i.e. your application namespace, *myapp* in the previous example. If null, the default namespace will be used, which is **ezsettings** by default)
3. `$scope` (i.e. a siteaccess name. If null, the current siteaccess will be used)

Inject the ConfigResolver in your services

You can use the **ConfigResolver** in your own services whenever needed. To do this, just inject the `ezpublish.config.resolver` service:

```

parameters:
    my_service.class: My\Cool\Service

services:
    my_service:
        class: %my_service.class%
        arguments: [@ezpublish.config.resolver]

```

```

<?php
namespace My\Cool;

use eZ\Publish\Core\MVC\ConfigResolverInterface;

class Service
{
    /**
     * @var \eZ\Publish\Core\MVC\ConfigResolverInterface
     */
    private $configResolver;

    public function __construct( ConfigResolverInterface $configResolver )
    {
        $this->configResolver = $configResolver;
        $myParam = $this->configResolver->getParameter( 'foo', 'myapp' );
    }

    // ...
}

```

Custom locale configuration (feature available since 5.1)

If you need to use a custom locale they can also be configurable in `ezpublish.yml`, adding them to the *conversion map*:

```

ezpublish:
    # Locale conversion map between eZ Publish format (i.e. fre-FR) to
    # POSIX (i.e. fr_FR).
    # The key is the eZ Publish locale. Check locale.yml in
    # EzPublishCoreBundle to see natively supported locales.
    locale_conversion:
        eng-DE: en_DE

```

A locale *conversion map* example can be found in the `core` bundle, on `locale.yml`.

- Legacy configuration
- Legacy configuration injection
- Logging configuration
- Persistence cache configuration
- View provider configuration