Legacy code and features

eZ Publish 5 has a strong focus on backwards compatibility and thus lets you reuse code you might have written for 4.x, including templates and modules.

> **Hint**
> Read Intro for eZ Publish 4.x/3.x developers to have an overview of common concepts and terminology changes, it also have a sub page comparing 4.x with 5.x.

> **Looking for 4.x documentation ?**
> You will find the 4.x documentation in the legacy part of the documentation.

- Legacy Mode
  - What it does
  - What it doesn't do
  - Allowing Symfony routes to work
- Legacy Template inclusion
  - Template parameters
- Running legacy code
- Legacy modules
  - Routing fallback & sub-requests
  - Using eZ Publish 5 and Symfony features in Legacy
  - Running legacy scripts and cronjobs
    - Script help
- Legacy bundles
  - Creating a legacy bundle extension
  - Alternative: referencing an existing legacy extension via composer
  - Running the legacy bundle install script manually

Legacy Mode

Legacy mode is a specific configuration mode where eZ Publish's behavior is the closest to v4.x. It might be used in some very specific use cases, such as **running the admin interface**.

What it does

- **Still runs through the whole Symfony kernel**. As such, Symfony services can still be accessed from legacy stack.
- **Disables the default router** (standard Symfony routes won't work in this mode)
- **Disables the UrlAliasRouter**. As such, the **ViewController will be bypassed**.

What it doesn't do

- **Increase performance**. Legacy mode is actually **painful for performances** since it won't use the HttpCache mechanism.

In a migration context, **using Legacy Mode is never a good option** as it prevents all the performance goodness (e.g. Http Cache) to work.
Always keep in mind that, **not running in legacy mode, if a content item still doesn't have a corresponding Twig template/controller, eZ Publish will always fallback to the legacy kernel, looking for a legacy template** .

> If you don't have anything (e.g. pagelayout, config...) migrated to Symfony stack, including Twig templates and Symfony controllers, **it's usually better performance wise to use *pure legacy stack*, using `ezpublish_legacy/` folder as your DocumentRoot**.
>
> This way you will only use the legacy kernel and eZ Publish will have the same behavior as 4.x. However, note that **you won't be able to use the Symfony stack features at all**.

Allowing Symfony routes to work

Symfony routes are disabled in legacy mode, which implies admin interface as well.

If for some reason you need a Symfony route to work, you add it to a whitelist :

```
ez_publish_legacy:
    # Routes that are allowed when legacy_mode is true.
    # Must be routes identifiers (e.g. "my_route_name").
    # Can be a prefix, so that all routes beginning with given prefix will be taken
into account.
    legacy_aware_routes: ["my_route_name", "my_route_prefix_"]
```

Deprecation warning
Configuration for legacy aware routes have changed as of 5.4.2. Prior to this version, configuration was the following:

```
ezpublish:
    router:
        default_router:
            # Routes that are allowed when legacy_mode is true.
            # Must be routes identifiers (e.g. "my_route_name").
            # Can be a prefix, so that all routes beginning with given prefix
will be taken into account.
            legacy_aware_routes: ["my_route_name", "my_route_prefix_"]
```

Note that this notation doesn't work as of eZ Platform (kernel v6.0)

By default, `_ezpublishLegacyTreeMenu` and all REST v2 (`ezpublish_rest_` prefix) routes are allowed.

Legacy Template inclusion

It is possible to include old templates (**.tpl**) into new ones.

Include a legacy template using the old template override mechanism

```
{# Twig template #}
{# Following code will include my/old_template.tpl, exposing $someVar variable in it
#}
{% include "design:my/old_template.tpl" with {"someVar": "someValue"} %}
```

Or if you want to **include a legacy template by its path**, relative to `ezpublish_legacy` folder:

eZ Publish 5.1+

```
{# Following code will include
ezpublish_legacyextension/my_legacy_extension/design/standard/templates/my_old_templat
e.tpl, exposing $someVar variable in it #}
{% include
"file:extension/my_legacy_extension/design/standard/templates/my_old_template.tpl"
with {"someVar": "someValue"} %}
```

Template parameters

Scalar and array parameters are passed to a legacy template *as-is*.

Objects, however, are being converted in order to comply the legacy eZ Template API. By default a generic adapter is used, exposing all public properties and getters. You can define your own converter by implementing the appropriate interface and declare it as a service with the **`ezpublish_legacy.templating.converter`** tag.

> `Content` / `Location` objects from the Public API are converted into `eZContentObject`/`eZContentObjectTreeNode` objects (re-fetched).

Running legacy code

eZ Publish 5 still relies on the legacy kernel (from 4.x) and runs it when needed **inside an isolated PHP closure**, making it **sandboxed**. This is available for your use as well through the `runCallback()` method.

> All calls from Platform/Symfony stack to legacy stack should be run inside a `runCallback()` call.

Simple legacy code example

```
<?php
// Declare use statements for the classes you may need
use eZINI;

// Inside a controller extending eZ\Bundle\EzPublishCoreBundle\Controller
$settingName = 'MySetting';
$test = array( 'oneValue', 'anotherValue' );
$myLegacySetting = $this->getLegacyKernel()->runCallback(
    function () use ( $settingName, $test )
    {
        // Here you can reuse $settingName and $test variables inside the legacy
context
        $ini = eZINI::instance( 'someconfig.ini' );
        return $ini->variable( 'SomeSection', $settingName );
    }
);
```

The example above is very simple and naive - in fact for accessing configuration settings from the Legacy Stack using the ConfigResolver is recommended.

Using the legacy closure, you'll be able to even run complex legacy features, like an **eZ Find search**:

Using eZ Find

```
use eZFunctionHandler;

$searchPhrase = 'My search phrase';
$sort = array(
    'score'     => 'desc',
    'published' => 'desc'
);
$contentTypeIdenfiers = array( 'folder', 'article' );
$mySearchResults = $this->getLegacyKernel()->runCallback(
    function () use ( $searchPhrase, $sort, $contentTypeIdenfiers )
    {
        // eZFunctionHandler::execute is the equivalent for a legacy template fetch
function
        // The following is the same than fetch( 'ezfind', 'search', hash(...) )
        return eZFunctionHandler::execute(
            'ezfind',
            'search',
            array(
                'query'     => $searchPhrase,
                'sort_by'   => $sort,
                'class_id'  => $contentTypeIdenfiers
            )
        );
    }
);
```

Legacy modules

Routing fallback & sub-requests

Any route that is not declared in eZ Publish 5 in an included `routing.yml` and that is not a valid *UrlAlias* **will automatically fallback to eZ Publish legacy** (including admin interface).

This allows all your old modules to work as before, out-of-the-box (including kernel modules), and also allows you to reuse this code from your templates using sub requests:

Template legacy module sub-request

```
{{ render( url( 'ez_legacy', {'module_uri': '/content/view/sitemap/2'} ) ) }}
```

If your module uses ezjscore to insert CSS or JS, you need to add calls to ez_legacy_render_js and/or ez_legacy_render_css to the twig template rendering the <head> of your page.

Using eZ Publish 5 and Symfony features in Legacy

If for some reason you need to develop a legacy module and access to eZ Publish 5 / Symfony features (i.e. when developing an extension for admin interface), you'll be happy to know that you actually have access to all services registered in the whole framework, including bundles, through the service container.

The example below shows how to retrieve the content repository and the logger.

Retrieve services from the container

```
// From a legacy module or any PHP code running in legacy context.
$container = ezpKernel::instance()->getServiceContainer();

/** @var $repository \eZ\Publish\API\Repository\Repository */
$repository = $container->get( 'ezpublish.api.repository' );
/** @var $logger
\Symfony\Component\HttpKernel\Log\LoggerInterface|\Psr\Log\LoggerInterface */
// PSR LoggerInterface is used in eZ Publish 5.1 / Symfony 2.2
$logger = $container->get( 'logger' );
```

Tip
The example above works in legacy modules and CLI scripts

Running legacy scripts and cronjobs

Note: This feature has been introduced in eZ Publish 5.1.

Important
Running legacy scripts and cronjobs through the Symfony stack is highly recommended !
Otherwise, features from the Symfony stack cannot be used (i.e. HTTP cache purge) and cache clearing. NB: Some script we know won't affect cache, are still documented to be executed the direct way.

Legacy scripts can be executed form the Symfony CLI, by using the `ezpublish:legacy:script` command, specifying the path to the script as argument.

The command will need to be executed from eZ Publish's 5 root, and the path to the desired script must exist in the `ezpublish_legacy` folder. Here's a usage example:

```
php ezpublish/console --env=prod ezpublish:legacy:script
bin/php/ezpgenerateautoloads.php
```

Here we made sure to specify --env=prod, this is needed for all legacy scripts that clear cache, otherwise they will will clear dev environment cache instead of prod for Symfony stack.

Options and arguments
Always pass the legacy script options and arguments **AFTER** script path, otherwise they will be lost.

Script help

If you want to access the script's help please be aware that you will need to use the newly introduced `--legacy-help` option, since --help is already reserved for the CLI help.

The `--legacy-help` option should be added before the path to the script for this to work.

Here's an example:

```
php ezpublish/console --env=prod ezpublish:legacy:script --legacy-help
bin/php/ezpgenerateautoloads.php
```

The same logic will apply for cronjob execution.
Legacy cronjobs are triggered by the `runcronjobs.php` legacy script, which expects the name of the cronjob to run as a parameter.
This is how you can run cronjobs from the Symfony CLI:

```
php ezpublish/console --env=prod ezpublish:legacy:script runcronjobs.php
frequent
```

Also, if you require using additional script options, please be sure to use the long name, such as `--siteaccess` or `--debug` to avoid conflicts between script and CLI options.
For more details regarding legacy cronjobs execution please refer to the Running cronjobs chapter existing in doc.ez.no.

Legacy bundles

> Available starting from v5.3 / 2014.03.

Most customization work on eZ Publish legacy was done through Extensions. Due to the current dual-kernel architecture, many features written for the new stack will require some matching legacy code (a FieldType will require the equivalent datatype, a feature might require back-office customization...). In order to facilitate this, legacy bundles were implemented.

They allow you to place a legacy extension (or several) within a Symfony 2 bundle. Any such extension will be installed inside `ezpublish_legacy/extension`, and automatically enabled as long as the bundle is registered.

Creating a legacy bundle extension

Example: https://github.com/ezsystems/CommentsBundle

Legacy extensions must:

- be placed within the bundle, within an `ezpublish_legacy` subfolder
- must be contained in their own subfolder: `Acme/AcmeBundle/ezpublish_legacy/acmeextension`
- must contain an `extension.xml` file

A symlink (by default) will be created in `ezpublish_legacy/extension` pointing to the `acmeextension` folder. Starting from there, it will behave like any regular legacy extension.

Alternative: referencing an existing legacy extension via composer

Example: https://github.com/ezsystems/ngsymfonytools-bundle.

An alternative use-case is also covered: you have an existing legacy extension, and a new stack bundle depends on it. It is possible to reference this legacy extension, without copying anything from it, and have it automatically installed and enabled when the bundle is installed and registered.

To do so, the bundle's Bundle class must implement an extra interface, `eZ\Bundle\EzPublishLegacyBundle\LegacyBundles\LegacyBundleInterface` . This interface specifies a `getLegacyExtensionsNames()` method, that is expected to return an array of legacy extensions names. Those legacy extension names will be enabled in legacy.

In ngsymfonytoolsbundle, we have two things:

- `composer.json` requires the legacy extension
  When the bundle is installed using composer, the legacy extension gets installed inside legacy
- EzSystemsNgsymfonytoolsBundle.php implements `getLegacyExtensionsNames()`, and returns `array( 'ngsymfonytools' )`, automatically enabling the extension in legacy.

Running the legacy bundle install script manually

By default, `ezpublish-community/composer.json` will call the legacy bundle install script after update and install. If for some reason, you want to do it manually, it looks a lot like asset install scripts:

```
php ezpublish/console ezpublish:legacybundles:install_extensions
```

By default, it will create an absolute symlink, but options exist to use a hard copy (`–copy`) or a relative link (`--relative`).
The script will also avoid overwriting existing targets if they aren't links to the bundle. The `--force` option will make the script *erase* existing targets before copying/linking.