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.

This page describes built in legacy "bridge" features in 5.x. If you are looking for the same component for eZ Platform, have a look at LegacyBridge which is a optional but newer version of what is described here

Hint
Read Architecture and 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 describing legacy itself in the legacy part of the documentation.

- Overview of ways to run 5.x
- 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

Overview of ways to run 5.x

5.x can be run in different ways as shown on Architecture page, here is the three main ways to run it, where the first two can be controlled in configuration and combined within the same installation:

	SiteAccess setting	Supported on 5.x	Supported in the future[1]	Allows use of Symfony & Platform features[2]
Legacy mode	enabled	yes	yes	yes[3]
	disabled	yes	yes	yes
Pure legacy	N/A	yes	no	no

Footnotes:

1. Supported in the future on eZ Platform 1.13/2.x, optionally also in combination with eZ Platform Cloud.
2. Allows use of features in Platform / Symfony including Legacy migration features described on this page.
3. One feature not being used in this mode is HttpCache, but all integrations to clear cache such as Http *(for other siteaccessse, rest, ..)* is still enabled, see below for further info.

Legacy Mode

Legacy mode, when enabled for a SiteAccess is a specific configuration mode where eZ Publish 5.x's behavior is the closest to v4.x. It might be used in some very specific use cases, such as **running the admin interface**, or as a temporary tool while migrating to Platform stack.

What it does

- **Still runs through the whole Symfony kernel**. As such, Symfony services can still be accessed from legacy stack, and all integrations

like session, cache and users works as intended.
- **Disables the default router,** standard Symfony routes won't work in this mode, unless allowing it explicitly.
- **Disables the UrlAliasRouter**, letting instead legacy handle url alias lookups**.** As such, the Platform stack **ViewController will be bypassed**.

What it doesn't do

- **Increase performance**. Legacy mode actually **degrades performances** since it won't use the HttpCache mechanism, but still needs to boot up Platform/Symfony stack and setup integrations before starting eZ Publish legacy.

How to avoid this
You have different options to avoid this:

1. **Migrate your pagelayout to Platform stack**, in order to be able to disable legacy mode and take advantage of HttpCache
   - In this setup, any kind of content type which is missing override rules in Platform stack, eZ Publish will always fallback to the legacy kernel, looking for a legacy template.
   - Same goes for custom legacy modules, if URL's are not handled by Symfony or by url alias, fallback goes to legacy for handling the url.

   However legacy admin will still be affected as it's in legacy mode, so the recommendation is to consider this in combination with next point.
2. **Upgrade to PHP 7.0**, in order to hide the performance degradation and as a bonus reduce memory by half and gain 1.5-2x performance on Symfony routes.
   This avoids editors complaining about the slower backend and allows you to gradually do step 1 across several SiteAccesses.
   This can be done one of two ways:
   a. *Very low effort:* Update to latest 5.4.x release in order to be able to use PHP 7.0 with a version of eZ Publish which has been made compatible with PHP7.
   b. *Low-Some effort:* Upgrade to eZ Platform v2, install the legacy bridge and continue to use legacy admin, get an eZ Publish legacy 2017/8.x release which supports PHP 7.1 100%, but for that you'll need to adapt your eZ Publish legacy PHP code a bit.
3. **Last resort: Point web server directly at eZ Publish legacy**, loose access to all Symfony/Platform/Legacy-bridge feature-set in order to postpone dealing with options above.
   In this setup you use `ezpublish_legacy/` folder as your `DocumentRoot`, meaning the installs will have 99.9% the same behavior as 4.x with except changes done to deprecated features.
   Recommended for: Large projects coming from 4.x and which first needs to bring legacy code base as a small effort up to 5.x legacy, before evaluating further steps either on 5.x or to eZ Platform with or without Legacy Bridge.
   Not recommended for: Future compatibility, as this is "**pure legacy**" setup *(see Architecture)* it will not be officially supported in the future on either eZ Platform or eZ Platform Cloud.

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_"]
```

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 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, otherwise any writes or reads to disk can fail.

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 instead.

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

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 directly.

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

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:

1. be placed within the bundle, within an `ezpublish_legacy` subfolder
2. must be contained in their own subfolder: `Acme/AcmeBundle/ezpublish_legacy/acmeextension`
3. 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:

1. `composer.json` requires the legacy extension
   When the bundle is installed using composer, the legacy extension gets installed inside legacy
2. 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.