Design

Introduction

This page covers design in eZ Platform in a general aspect. If you want to learn about how to display content and build your content templates, you might want to check Content Rendering.

To apply a template to any part of your webpage, you need three (optionally four) elements:

1. An entry in the configuration that defines which template should be used in what situation
2. The template file itself
3. Assets used by the template (for example, CSS or JS files, images, etc.)
4. (optional) A custom controller used when the template is read which allows you more detailed control over the page.

Configuration

Each template must be mentioned in a configuration file together with a definition of the situation in which it is used. You can use the `ezplatform.yml` file located in the `app/config/` folder, or create your own separate configuration file in that folder that will list all your templates.

If you decide to create a new configuration file, you will need to import it by including an import statement in `ezplatform.yml`. Add the following code at the beginning of `ezplatform.yml`:

```
imports:
    - { resource: <your_file_name>.yml }
```

If you are using the recommended .yml files for configuration, here are the basic rules for this format:

The configuration is based on pairs of a key and its value, separated by a colon, presented in the following form: key: value. The value of the key may contain further keys, with their values containing further keys, and so on. This hierarchy is marked using indentation – each level lower in the hierarchy must be indented in comparison with its parent.

A short configuration file can look like this:

In this topic:

Sample configuration file

```
ezpublish:
    system:
        default:
            user:
                layout: pagelayout.html.twig
            content_view:
                full:
                    article:
                        template: full\article.html.twig
                        match:
                            Identifier\ContentType:
[article]
                    blog_post:
                        controller:
app.controller.blog:showBlogPostAction
                        template:
full\blog_post.html.twig
                        match:
                            Identifier\ContentType:
[blog_post]
                line:
                    article:
                        template: line\article.html.twig
                        match:
                            Identifier\ContentType:
[article]
```

This is what individual keys in the configuration mean:

- **`ezpublish`** and **`system`** are obligatory at the start of any configuration file which defines views.
- **`default`** defines the siteaccess for which the configuration will be used. "default", as the name suggests, determines what views are used when no other configuration is chosen. You can also have separate keys defining views for other siteaccesses.
- **`user`** and **`layout`** point to the main template file that is used in any situation where no other template is defined. All other templates extend this one. See below for more information.
- **`content_view`** defines the view provider.

> In earlier versions of eZ CMS, `location_view` was used as the view provider. It is now deprecated.

- **`full`** and **`line`** determine the kind of view to be used (see below).
- **`article`** and **`blog_post`** are the keys that start the configuration for one individual case of using a template. You can name these keys any way you want, and you can have as many of them as you need.
- **`template`** names the template to be used in this case, including the folder it is stored in (starting from `app/Resources/views`).
- **`controller`** defines the controller to be used in this case. Optional, if this key is absent, the default controller is used.
- **`match`** defines the situation in which the template will be used. There are different criteria which can be used to "match" a template to a situation, for example a Content Type, a specific Location ID, Section, etc. You can view the full list of matchers here: View provider configuration . You can specify more than one matcher for any template; the matchers will be linked with an AND operator.

In the example above, three different templates are mentioned, two to be used in full view, and one

in line view. Notice that two separate templates are defined for the "article" Content Type. They use the same matcher, but will be used in different situations – one when an Article is displayed in full view, and one in line view. Their templates are located in different folders. The line template will also make use of a custom controller, while the remaining cases will employ the default one.

> **Full, line and other views**
>
> Each Content item can be rendered differently, using different templates, depending on the type of view it is displayed in. The default, built-in views are **full** (used when the Content item is displayed by itself, as a full page), **line** (used when it is displayed as an item in the list, for example a listing of contents of a folder), and **embed** (used when one Content item is embedded in another). Other, custom view types can be created, but only these three have built-in controllers in the system.

> See View provider configuration for more details.

Template file

Templates in eZ Platform are written in the Twig templating language.

> **Twig templates in short**
>
> At its core, a Twig template is an HTML frame of the page that will be displayed. Inside this frame you define places (and manners) in which different parts of your Content items will be displayed (rendered).
>
> Most of a Twig template file can look like an ordinary HTML file. This is also where you can define places where Content items or their fields will be embedded.

The configuration described above lets you select one template to be used in a given situation, but this does not mean you are limited to only one file per case. It is possible to include other templates in the main template file. For example, you can have a single template for the footer of a page and include it in many other templates. Such templates do not need to be mentioned in the configuration .yml file.

> See Including Templates in Symfony documentation for more information on including templates.

The main template for your webpage (defined per siteaccess) is placed in the `pagelayout.html.twig` file. This template will be used by default for those parts of the website where no other templates are defined.

A `pagelayout.html.twig` file exists already in Demo Bundles, but if you are using a clean installation, you need to create it from scratch. This file is typically located in a bundle, for example using the built-in AppBundle: `src/AppBundle/Resources/views`. The name of the bundle must the added whenever the file is called, like in the example below.

Any further templates will extend and modify this one, so they need to start with a line like this:

```
{% extends "AppBundle::pagelayout.html.twig" %}
```

> Although using AppBundle is recommended, you could also place the template files directly in `<installation_folder>/app /Resources/views` . Then the files could be referenced in code without any prefix. See Best Practices for more information.

Template paths
In short, the `Resources/views` part of the path is automatically added whenever a template file is referenced. What you need to provide is the bundle name, name of any subfolder within `/views/`, and file name, all three separated by colons (:)

To find out more about the way of referencing template files placed in bundles, see Referencing Templates in a Bundle in Symfony documentation.

Templates can be extended using a Twig `block` tag. This tag lets you define a named section in the template that will be filled in by the child template. For example, you can define a "title" block in the main template. Any child template that extends it can also contain a "title" block. In this case the contents of the block from the child template will be placed inside this block in the parent template (and override what was inside this block):

pagelayout.html.twig

```
{# ... #}
    <body>
        {% block title %}
            <h1>Default title</h1>
        {% endblock %}
    </body>
{# ... #}
```

child.html.twig

```
{% extends "AppBundle::pagelayout.html.twig" %}
{% block title %}
    <h1>Specific title</h1>
{% endblock %}
```

In the simplified example above, when the `child.html.twig` template is used, the "title" block from it will be placed in and will override the "title" block from the main template – so "Specific title" will be displayed instead of "Default title."

Alternatively, you can place templates inside one another using the `include` function.

See http://twig.sensiolabs.org/doc/templates.html# for detailed documentation on how to use Twig.

Embed content in templates

Now that you know how to create a general layout with Twig templates, let's take a look at the ways in which you can render content inside them.

There are several ways of placing Content items or their Fields inside a template. You can do it using one of the Twig functions described in detail here .

As an example, let's look at one of those functions: **ez_render_field** . It renders one selected Field of the Content item. In its simplest form this function can look like this:

```
{{ ez_render_field( content, 'description' ) }}
```

This renders the value of the Field with identifier "description" of the current Content item (signified by "content"). You can additionally choose a special template to be used for this particular Field:

```
{{ ez_render_field(
       content,
       'description',
       { 'template':
'AppBundle:fields:description.html.twig' }
   ) }}
```

> As you can see in the case above, templates can be created not only for whole pages, but also for individual Fields.

Another way of embedding Content items is using the **render_esi** function (which is not an eZ-specific function, but a Symfony standard). This function lets you easily select a different Content item and embed it in the current page. This can be used, for instance, if you want to list the children of a Content item in its parent.

```
{{ render_esi(controller('ez_content:viewAction',
{locationId: 33, viewType: 'line'} )) }}
```

This example renders the Content item with Location ID 33 using the line view. To do this, the function applies the 'ez_content:viewAction' controller. This is the default controller for rendering content, but can be substituted here with any custom controller of your choice.

Assets

Asset files such as CSS stylesheets, JS scripts or image files can be defined in the templates and need to be included in the directory structure in the same way as with any other web project. Assets are placed in the `web/` folder in your installation.

Instead of linking to stylesheets or embedding images like usually, you can use the `asset` function.

Controller

While it is absolutely possible to template a whole website using only Twig, a custom PHP controller gives many more options of customizing the behavior of the pages.

See Custom controllers for more information.

Usage

Creating a new design using Bundle Inheritance

Due to the fact that eZ Platform is built using the Symfony 2 framework, it is possible to benefit from most of its stock features such as Bundle Inheritance. To learn more about this concept in general, check out the related Symfony documentation.

Bundle Inheritance allows you to customize a template from a parent bundle. This is very convenient when creating a custom design for an already existing piece of code.

The following example shows how to create a customized version of a template from the DemoBundle.

Creating a bundle

Create a new bundle to host your design using the dedicated command (from your app installation):

```
php app/console generate:bundle
```

Configuring bundle to inherit from another

Following the related Symfony documentation, modify your bundle to make it inherit from the "eZDemoBundle". Then copy a template from the DemoBundle in the new bundle, following the same directory structure. Customize this template, clear application caches and reload the page. You custom design should be available.

Known limitation

If you are experiencing problems with routes not working after adding your bundle, take a look at thi s issue.

Reference

Twig Helper

eZ Platform comes with a Twig helper as a global variable named **`ezpublish`** .

This helper is accessible from all Twig templates and allows you to easily retrieve useful information.

Property	Description
`ezpublish.siteaccess`	Returns the current siteaccess.
`ezpublish.rootLocation`	Returns the root Location object
`ezpublish.requestedUriString`	Returns the requested URI string (also known as semanticPathInfo).
`ezpublish.systemUriString`	Returns the "system" URI string. System URI is the URI for internal content controller. If current route is not an URLAlias, then the current Pathinfo is returned.
`ezpublish.viewParameters`	Returns the view parameters as a hash.
`ezpublish.viewParametersString`	Returns the view parameters as a string.
`ezpublish.legacy`	Returns legacy information.
`ezpublish.translationSiteAccess`	Returns the translation SiteAccess for a given language, or null if it cannot be found.
`ezpublish.availableLanguages`	Returns the list of available languages.
`ezpublish.configResolver`	Returns the config resolver.

Legacy

> `ezpublish.legacy` is only available **when viewing content in legacy fallback** (e.g. no corresponding Twig templates)

The `ezpublish.legacy` property returns an object of type ParameterBag, which is a container for key/value pairs, and contains additional properties to retrieve/handle legacy information.

Property	Description

`ezpublish.legacy.all`	Returns all the parameters, with all the contained information.
`ezpublish.legacy.keys`	Returns the parameter keys only.
`ezpublish.legacy.get`	Returns a parameter by name.
`ezpublish.legacy.has`	Returns true if the parameter is defined.

Listing the available parameters

You can list the available parameters in `ezpublish.legacy` by using the `ezpublish.legacy.keys` property, as shown in the following example:

Example on retrieving the available parameters under ezpublish.legacy

```
{{ dump(ezpublish.legacy.keys()) }}
```

which will give a result similar to:

```
array
  0 => string 'view_parameters' (length=15)
  1 => string 'path' (length=4)
  2 => string 'title_path' (length=10)
  3 => string 'section_id' (length=10)
  4 => string 'node_id' (length=7)
  5 => string 'navigation_part' (length=15)
  6 => string 'content_info' (length=12)
  7 => string 'template_list' (length=13)
  8 => string 'cache_ttl' (length=9)
  9 => string 'is_default_navigation_part'
(length=26)
  10 => string 'css_files' (length=9)
  11 => string 'js_files' (length=8)
  12 => string 'css_files_configured'
(length=20)
  13 => string 'js_files_configured'
(length=19)
```

Retrieving legacy information

Legacy information is accessible by using the `ezpublish.legacy.get` property, which will allow you to access all data contained in $module_result, from the legacy kernel.

This allows you to import information directly into twig templates. For more details please check the available examples on using the `ezpublish.legacy.get` property for retrieving persistent variables and assets.

As a usage example, if you want to access the legacy information related to 'content_info' you can do it, as shown in the following example:

Example on accessing 'content_info' under ezpublish.legacy

```
{{ ezpublish.legacy.get('content_info') }}
```

The previous call will return the contents on the 'content_info' as an `array`, and if we `dump` it the result will be similar to the following:

```
array
  'object_id' => string '57' (length=2)
  'node_id' => string '2' (length=1)
  'parent_node_id' => string '1' (length=1)
  'class_id' => string '23' (length=2)
  'class_identifier' => string 'landing_page'
(length=12)
  'remote_id' => string
'8a9c9c761004866fb458d89910f52bee' (length=32)
  'node_remote_id' => string
'f3e90596361e31d496d4026eb624c983' (length=32)
  'offset' => boolean false
  'viewmode' => string 'full' (length=4)
  'navigation_part_identifier' => string
'ezcontentnavigationpart' (length=23)
  'node_depth' => string '1' (length=1)
  'url_alias' => string '' (length=0)
  'current_language' => string 'eng-GB'
(length=6)
  'language_mask' => string '3' (length=1)
  'main_node_id' => string '2' (length=1)
  'main_node_url_alias' => boolean false
  'persistent_variable' =>
    array
      'css_files' =>
        array
          0 => string 'video.css' (length=9)
      'js_files' =>
        array
          0 => string 'video.js' (length=8)
  'class_group' => boolean false
  'state' =>
    array
      2 => string '1' (length=1)
  'state_identifier' =>
    array
      0 => string 'ez_lock/not_locked'
(length=18)
  'parent_class_id' => string '1' (length=1)
  'parent_class_identifier' => string 'folder'
(length=6)
  'parent_node_remote_id' => string
'629709ba256fe317c3ddcee35453a96a' (length=32)
  'parent_object_remote_id' => string
'e5c9db64baadb82ab8db54f0e2192ec3' (length=32)
```

Additionally, for retrieving information contained in 'content_info' such as the current language of the content in the page you can do it like in the following example:

Example on retrieving 'current_language'

```
{{
ezpublish.legacy.get('content_info')['current_language']
}}
```