Content view

REWRITE

The ViewController

eZ Platform comes with a native controller to display your content, known as the **`ViewController`**. It is called each time you try to reach a Content item from its **Url Alias** (human readable, translatable URI generated for any content based on URL patterns defined per Content Type) and is able to render any content previously edited in the admin interface or via the PHP API Tutorials.

It can also be called straight by its direct URI : `/content/view/<contentId>/<languageCode>`

A Content item can also have different **view types** (full page, abstract in a list, block in a landing page, etc.). By default the view type is **full** (for full page), but it can be anything (*line*, *block, etc*.).

Important note regarding visibility
The Location visibility flag, which you can change by hiding/revealing in the Platform UI, is not permission-based and thus acts as a simple potential filter. **It is not meant to restrict access to content**.

If you need to restrict access to a given Content item, use **Sections** or **Object states**, which are permission-based.

View selection

To display a Content item, the ViewController uses a view manager which selects the appropriate template depending on matching rules.

For more information about the **view provider configuration**, please refer to the dedicated page.

You can also use your own custom controller to render a content item/location.

Content view templates

A content view template is like any other template, with several specific aspects.

Available variables

Variable name	Type	Description
`location`	eZ\Publish\Core\Repository\Values\Content\Location	The Location object. Contains meta information on the content (ContentInfo) (only when accessing a Location)
`content`	eZ\Publish\Core\Repository\Values\Content\Content	The Content item, containing all Fields and version information (VersionInfo)
`noLayout`	Boolean	If true, indicates if the Content item/Location is to be displayed without any pagelayout (i.e. AJAX, sub-requests, etc.). It's generally `false` when displaying a Content item in view type **full**.
`viewBaseLayout`	String	The base layout template to use when the view is requested to be generated outside of the pagelayout (when `noLayout` is true).

Template inheritance and sub-requests

Like any template, a content view template can use template inheritance. However keep in mind that your content may be also requested via sub-requests (see below how to render embedded content objects), in which case you probably don't want the global layout to be used.

If you use different templates for embedded content views, this should not be a problem. If you'd rather use the same template, you can use an extra `noLayout` view parameter for the sub-request, and conditionally extend an empty pagelayout:

```
{% extends noLayout ? viewbaseLayout : "AcmeDemoBundle::pagelayout.html.twig" %}

{% block content %}
...
{% endblock %}
```

Rendering Content item's Fields

As stated above, a view template receives the requested Content item, holding all Fields.

In order to display the Fields' value the way you want, you can either manipulate the Field Value object itself, or use a custom template.

Getting raw Field value

Having access to the Content item in the template, you can use its public methods to access all the information you need. You can also use the ez_field_value helper to get the Field value. It will return the correct language if there are several, based on language priorities.

```
{# With the following, myFieldValue will be in the content's main language, regardless
of the current language #}
{% set myFieldValue = content.getFieldValue( 'some_field_identifier' ) %}

{# Here myTranslatedFieldValue will be in the current language if a translation is
available. If not, the content's main language will be used #}
{% set myTranslatedFieldValue = ez_field_value( content, 'some_field_identifier' ) %}
```

Using the Field Type's template block

All built-in Field Types come with their own Twig template. You can render any Field using this default template using the `ez_render_field()` helper.

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

Refer to `ez_render_field()` reference page for further information.

As this makes use of reusable templates, **using `ez_render_field()` is the recommended way and is to be considered the best practice**.

Rendering Content name

The **name** of a Content item is its generic "title", generated by the repository based on the Content Type's naming pattern. It often takes the form of a normalized value of the first field, but might be a concatenation of several fields. There are 2 different ways to access to this special property:

- Through the name property of ContentInfo (not translated).
- Through VersionInfo with the TranslationHelper (translated).

Translated name

The *translated name* is held in a `VersionInfo` object, in the names property which consists of hash indexed by locale. You can easily retrieve it in the right language via the `TranslationHelper` service.

```
<h2>Translated content name: {{ ez_content_name( content ) }}</h2>
<h3>Also works from ContentInfo : {{ ez_content_name( content.contentInfo ) }}</h3>
```

The helper will by default follow the prioritized languages order. If there is no translation for your prioritized languages, the helper will always return the name in the main language.

You can also **force a locale** in a second argument:

```
{# Force fre-FR locale. #}
<h2>{{ ez_content_name( content, 'fre-FR' ) }}</h2>
```

You can refer to ez_content_name() reference page for further information.

Name property in ContentInfo

This property is the actual content name, but **in main language only** (so it is not translated).

```
<h2>Content name: {{ content.contentInfo.name }}</h2>
```

```
$contentName = $content->contentInfo->name;
```

Exposing additional variables

It is possible to expose additional variables in a content view template. See parameters injection in content views or using your own custom controller to render a content item/location.

Making links to other locations

Linking to other locations is fairly easy and is done with a native `path()` Twig helper (or `url()` if you want to generate absolute URLs). You just have to pass it the Location object and `path()` will generate the URLAlias for you.

```
{# Assuming "location" variable is a valid
eZ\Publish\API\Repository\Values\Content\Location object #}
<a href="{{ path( location ) }}">Some link to a location</a>
```

If you don't have the Location object, but only its ID, you can generate the URLAlias the following way:

```
<a href="{{ path( "ez_urlalias", {"locationId": 123} ) }}">Some link to a location,
with its Id only</a>
```

You can also use the Content ID. In that case the generated link will point to the Content item's main Location.

```
<a href="{{ path( "ez_urlalias", {"contentId": 456} ) }}">Some link from a
contentId</a>
```

Under the hood
In the backend, `path()` uses the Router to generate links.

This makes it also easy to generate links from PHP, via the `router` service.

See also: Cross-siteaccess links

Render embedded content objects

Rendering an embedded content from a Twig template is pretty straight forward as you just need to **do a subrequest with `ez_content` controller**.

Using `ez_content` controller

This controller is exactly the same as the ViewController presented above. It has one main `viewAction`, that renders a Content item.

You can use this controller from templates with the following syntax:

```
{{ render( controller( "ez_content:viewAction", {"contentId": 123, "viewType": "line"}
) ) }}
```

The example above renders the Content whose ID is **123**, with the view type **line**.

Reference of `ez_content` controller follows the syntax of *controllers as a service*, as explained in Symfony documentation.

Available arguments

As with any controller, you can pass arguments to `ez_content:viewLocation` or `ez_content:viewContent` to fit your needs.

Name	Description	Type	Default value
`contentId`	ID of the Content item you want to render. **Only for `ez_content:viewContent`**	integer	N/A

`locationId`	ID of the Location you want to render. **Only for `ez_content:viewLocation`**	integer	Content item's main location, if defined
`viewType`	The view type you want to render your Content item/Location in. Will be used by the ViewManager to select a corresponding template, according to defined rules. Example: full, line, my_custom_view, etc.	string	full
`layout`	Indicates if the sub-view needs to use the main layout (see available variables in a view template)	boolean	false

`params`	Hash of variables you want to inject to sub-template, key being the exposed variable name.
	`{{ render(`
	`      controller(`
	`          "ez_content:viewAction",`
	`          {`
	`              "contentId": 123,`
	`              "viewType": "line",`
	`              "params": { "some_variable":`
	`"some_value" }`
	`          }`
	`      )`
	`) }}`
		hash	empty hash

Render block

You can specify which controller will be called for a specific block view match, much like defining custom controllers for location view or content view match.

Also, since there are two possible actions with which one can view a block: `ez_page:viewBlock` and `ez_page:viewBlockById`, it is possible to specify a controller action with a signature matching either one of the original actions.

Example of configuration in `app/config/ezplatform.yml`:

```
ezpublish:
    system:
        eng_frontend_group:
            block_view:
                ContentGrid:
                    template: NetgenSiteBundle:block:content_grid.html.twig
                    controller: NetgenSiteBundle:Block:viewContentGridBlock
                    match:
                        Type: ContentGrid
```

ESI

Just as for regular Symfony controllers, you can take advantage of ESI and use different cache levels:

Using ESI

```
{{ render_esi( controller( "ez_content:viewAction", {"contentId": 123, "viewType":
"line"} ) ) }}
```

Only scalable variables can be sent via render_esi (not object)

Asynchronous rendering

Symfony also supports asynchronous content rendering with the help of hinclude.js library.

Asynchronous rendering

```
{{ render_hinclude( controller( "ez_content:viewAction", {"contentId": 123,
"viewType": "line"} ) ) }}
```

Only scalable variables can be sent via render_hinclude (not object)

Display a default text

If you want to display a default text while a controller is loaded asynchronously, you have to pass a second parameter to your render_hinclude twig function.

Display a default text during asynchronous loading of a controller

```
{{ render_hinclude( controller( 'EzCorporateDesignBundle:Header:userLinks' ), {
'default': "<div style='color:red'>loading</div>" }) }}
```

See also: How to use a custom controller to display a content item or location

hinclude.js needs to be properly included in your layout to work.

Please refer to Symfony documentation for all available options.

Related topics:

- View provider configuration
- Default view templates
- Parameters injection in content views
- How to use a custom controller to display a content item or location