The Page Field Type

Deprecated
The Page Field Type has as of first stable releases of eZ Platform 15.12 and eZ Studio 15.12 been deprecated. A replacement called L andingPage is provided with eZ Studio and tools for migration are planned to be provided together with 16.02/16.04 release. At that point Page Field Type will not be bundled anymore.

Description

With the Page Field Type, in legacy part of "ezflow" extension, editors can define a **layout** with **multiple zones** within a single front- or landing-page.

Within each zone, editors create **blocks** that contain particular content categories. Specific content can be added to these blocks, they are called **block items**.

This is particularly useful for managing homepages/landing pages.

The Page Field Type has limited write support with the Public API as of **EZP-25413** - Page FieldType Public API should save custom attributes and rotation **CLOSED** with 16.04 *(v1.3.0)* release, before this it where read-only. In all versions it is still possible to edit content with it through the admin interface *(which runs through the legacy stack)*.

Name	Internal name	Expected input	Output
`Page`	`ezpage`	`N/A`	`eZ\Publish\Core\FieldType\Page\Parts\Page`

Configuration

Warning
You still need to define your available layouts and blocks in the legacy part to get them available in the admin interface. Please refer to eZ Publish legacy documentation to learn how to do so.

Defining a zone layout

A layout is a combination of zones that are placed on a page. The placement of the zones is defined in a template that is specified as part of the layout configuration. You can define as many layouts as you need.

You can define a new layout and enable it in your main YAML configuration:

ezplatform.yml

```
ezpublish:
    system:
        my_siteaccess:
            ezpage:
                layouts:
                    myLayoutIdentifier:
                        name: "My über cool layout"
                        template:
"AcmeDemoBundle:page/zonelayouts:my_template.html.twig"
    enabledLayouts: [myLayoutIdentifier]
```

Then, when rendering a Page Field Type using `myLayoutIdentifier`, `Resources/views/page/zonelayouts/my_template.html.twig` from AcmeDemoBundle will be used (see how to use template identifiers in Symfony documentation).

Tip
You can specify a legacy template in your layout definition.

```
ezpublish:
    system:
        my_siteaccess:
            ezpage:
                layouts:
                    myLegacyLayout:
                        name: My legacy layout
                        template: "design:zone/my_legacy_template.tpl"
    enabledLayouts: [myLayoutIdentifier]
```

However, doing so will defer block display to the legacy templates as well.

Available blocks

The blocks need to be defined and enabled in the YAML configuration as well:

ezplatform.yml

```
ezpublish:
    system:
        my_siteaccess:
            ezpage:
                blocks:
                    myBlockIdentifier:
                        name: "My über cool block"
                    myBlockIdentifier2:
                        name: "My über cool block 2"
    enabledBlocks: [myBlockIdentifier, myBlockIdentifier2]
```

Block template selection

Template selection rules are applied only when you render a block with the `PageController` (using `ez_page:viewBlock` from templates), see below.

Like you are able to define template selection rules when displaying Location and Content objects, you can also define rules for blocks, with dedicated matchers.

Configuration is a hash built in the following way:

ezplatform.yml

```
ezpublish:
    system:
        my_siteaccess:
            block_view:
                # A simple unique key for your matching ruleset
                my_rule_set:
                    # The template identifier to load, following the Symfony bundle
notation for templates
                    template: AcmeTestBundle:block:campaign.html.twig
                    # Hash of matchers to use, with their corresponding values to
match against
                    match:
                        # Key is the matcher "identifier" (class name or service
identifier)
                        # Value will be passed to the matcher's setMatchingConfig()
method.
                        Type: Campaign
                another_rule:
                    template: AcmeTestBundle:block:custom_block.html.twig
                    match:
                        Type: CustomBlock
```

Tip
You can define your template selection rules in a different configuration file. Read the cookbook recipe to learn more about it .

Matchers for `block_view` follow the same behavior than matchers for regular location_view / content_view, except that their relative namespace will be `eZ\Publish\Core\MVC\Symfony\View\BlockViewProvider\Configured\Matcher` .

Hence you can combine matchers with AND and OR capabilities (see main matchers' documentation page).

Available matchers

Identifier	Description

`Type`	Matches the unique **block identifier** defined in the legacy `block.ini` file (see legacy documentation).
	
	For example with the following configuration in legacy block.ini, it will match against `Manual3Items`:
	
	`[Manual3Items]`
	`Name=3 Column News`

`View`	Matches the **view's unique identifier** defined in the block definition in the legacy `block.ini` (see legacy documentation).
	For example with the following configuration in legacy `block.ini`, it will match against `3_items1`:
	`[Manual3Items]`
	`Name=3 Column News`
	`ViewList[]=3_items1`
	When no view is defined, the default value is **default**.

`Id\Block`	Matches against the block ID, as stored in `ezm_block` table
`Id\Zone`	Matches against the zone ID a block belongs to, as stored in `ezm_block` table

Displaying the Page content

This section focuses on how to display blocks from zone/layout templates.

Render of these templates are triggered when using `ez_render_field()` helper, like for any other field type.

See field rendering documentation for more information.

Layout template

Goal of a **layout** template is to display **zones** for the given layout, depending on your layout configuration.

Variables passed to the layout template

Variable name	Description	Type
`zones`	Zone objects for this Page field	Array of `eZ\Publish\Core\FieldType\Page\Parts\Zone` objects
`zone_layout`	The layout identifier (e.g. "2ZonesLayout1")	`string`
`pageService`	The PageService object (read more below).	`eZ\Bundle\EzPublishCoreBundle\FieldType\Page`

Rendering blocks

Each zone contain blocks that hold your content as block items. To render blocks from a layout template, you need to do a sub-request.

Tip
You can use a custom controller to display a block.

However, if you do so, you might need to get access to the PageService. You can get it via the service container with identifier `ezpublish.fieldType.ezpage.pageService`.

Using `ez_page:viewBlock`

This controller is responsible of choosing the right template for your block, depending on the rules you defined.

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

```
{{ render( controller( "ez_page:viewBlock", {'block': myBlock} ) ) }}
```

Available arguments

Name	Description	Type	Default value
`block`	The block object you want to render	`eZ\Publish\Core\FieldType\Page\Parts\Block`	N/A

`params`	Hash of variables you want to inject to sub-template, key being the exposed variable name.
	`{{ render(`
	`      controller(`
	`          "ez_page:viewBlock",`
	`          {`
	`              'block': myBlock,`
	`              'params': {`
	`'some_variable': 'some_value' }`
	`          }`
	`      )`
	`) }}`
		hash	empty

`cacheSettings`	Hash of cache settings to use by the sub-controller (useful if you use ESI or Hinclude strategies).
	`{{ render_esi(`
	`      controller(`
	`          "ez_page:viewBlock",`
	`          {`
	`              'block': myBlock,`
	`              'params': {`
	`'some_variable': 'some_value' },`
	`              'cacheSettings': {`
	`'smax-age': 600 }`
	`          }`
	`      )`
	`) }}`
		hash (accepted keys are `max-age` and `smax-age`)	empty

Variables exposed to the block template

Variable name	Type	Description
`block`	`eZ\Publish\Core\FieldType\Page\Parts\Block`	The block to display
`valid_items`	Array of `eZ\Publish\Core\FieldType\Page\Parts`	Displayable block items
`valid_contentinfo_items`	Array of `eZ\Publish\API\Repository\Values\Content\ContentInfo`	Displayable block items, as `ContentInfo` objects.
`pageService` (deprecated as of v5.2)	`eZ\Bundle\EzPublishCoreBundle\FieldType\Page\PageService`	The PageService object (deprecated)

And of course, all the additional variables you injected in the `params` argument .

> `valid_items` and `valid_contentinfo_items` variables are available as of **v5.2 / 2013.11**.
>
> Usage of `pageService` is deprecated as of **v5.2 / 2013.11.**

Using `pageView:viewBlockById`

`>= EZP 5.3.2`

You can render Blocks with ESI strategy in all contexts, ie including using Varnish with the controller viewBlockById.

> This feature is available from eZ Platform 5.3.2.

Available arguments

Name	Description	Type	Default value
`id`	The block ID of the block you want to render A `\eZ\Publish\API\Repository\Exceptions\NotFoundException` will be thrown, if block could not be found	`string`	N/A
`params`	Hash of variables you want to inject to sub-template, key being the exposed variable name. `{{ render( controller( "ez_page:viewBlockById", { 'id': 42, 'params': { 'some_variable': 'some_value' } } ) ) }}`	hash	empty

```
{{ render(
      controller(
          "ez_page:viewBlockById",
          {
              'id': 42,
              'params': { 'some_variable':
'some_value' }
          }
      )
) }}
```

`cacheSettings`	Hash of cache settings to use by the sub-controller (useful if you use ESI or Hinclude strategies). (see code below)	hash (accepted keys are `max-age` and `smax-age`)	empty

```
{{ render_esi(
      controller(
          "ez_page:viewBlockById",
          {
              'id': 42,
              'params': { 'some_variable':
'some_value' },
              'cacheSettings': {
'smax-age': 600 }
          }
      )
) }}
```

Rendering Block items

As said above, **a block holds your displayable content as block items** which consists of `eZ\Publish\Core\FieldType\Page\Parts\Item` objects. Among the available properties, you will find `contentId` and `locationId` which reference the content/location you want to display. All you have to do then is to **render it view `ez_content:viewLocation` or `ez_content:viewContent`** (see full example below).

The PageService object

The PageService object (`eZ\Bundle\EzPublishCoreBundle\FieldType\Page\PageService`) is a helper giving the possibility to get current zone/block definitions and to retrieve block items.

Main methods

Method name	Description	Return type
`getZoneDefinition()`	Returns zone definition (all defined zones for the current siteaccess) as an array	`array`
`getZoneDefinitionByLayout()`	Returns a zone definition for a given layout. It consists of a configuration array for the given layout.	`array`
`getBlockDefinition()`	Returns block definition as an array	`array`
`getBlockDefinitionByIdentifier()`	Returns a block definition for a given block identifier.	`array`
`getValidBlockItems()`	Returns valid items (that are to be displayed), for a given block.	`eZ\Publish\Core\FieldType\Page\Parts\Item[]`

`getLastValidBlockItem()`	Returns the last valid item, for a given block.	`eZ\Publish\Core\FieldType\Page\Parts\Item|null`
`getWaitingBlockItems()`	Returns queued items (the next to be displayed), for a given block.	`eZ\Publish\Core\FieldType\Page\Parts\Item[]`
`getArchivedBlockItems()`	Returns archived items (that were previously displayed), for a given block.	`eZ\Publish\Core\FieldType\Page\Parts\Item[]`
`getValidBlockItemsAsContentInfo()`	Returns valid block items as content objects	`eZ\Publish\API\Repository\Values\Content\ContentInfo[]`

Example

2zoneslayout1.html.twig

```
<h2>Twig Template for 2 zoneslayout1 zone</h2>
<div class="zone-layout-{{ zone_layout|lower }} row">
    <div class="span8">
        <section class="content-view-block">
        {% if zones[0].blocks %}
            {# Rendering blocks with default PageController #}
            {% for block in zones[0].blocks %}
                {{ render( controller( "ez_page:viewBlock", {'block': block} ) ) }}
            {% endfor %}
            <div class="block-separator"></div>
        {% endif %}
        </section>
    </div>
    <div class="span4">
        <aside>
            <section class="content-view-block content-view-aside">
            {% if zones[1].blocks %}
                {# Still rendering with default PageController, but passing specific
cache value (TTL of 100 seconds) and using ESI #}
                {% for block in zones[1].blocks %}
                    {{ render_esi( controller( "ez_page:viewBlock", {'block': block,
'cacheSettings': {'smax-age': 100}} ) ) }}
                {% endfor %}
                <div class="block-separator"></div>
            {% endif %}
            </section>
        </aside>
    </div>
</div>
```

campaign_block.html.twig

```
<h3>Twig Template for Campaign Block type</h3>
{% set validContentInfoItems = pageService.getValidBlockItemsAsContentInfo( block ) %}
{% set validItems = pageService.getValidBlockItems( block ) %}
<!-- BLOCK: START -->
<div class="block-type-campaign">
    <div class="campaign">
        <a href="#" class="navig prev" style="opacity:0;"><span
class="hide">&lt;</span></a>
        <a href="#" class="navig next"><span class="hide">&gt;</span></a>
        <ul class="indicator">
            {% for contentInfo in validContentInfoItems %}
                <li><span>{{ contentInfo.name }} (#{{ contentInfo.id }})</span></li>
            {% endfor %}
        </ul>
        <ul class="images">
        {# Rendering valid items with regular view controller, with
"block_item_campaign" view type #}
        {# Also passing an "image_class" parameter which will be available in
sub-template. #}
        {% for item in validItems %}
            {{ render(
                controller(
                    'ez_content:viewLocation',
                    {
                        'locationId': item.locationId,
                        'viewType': 'block_item_campaign',
                        'params': {'image_class': 'campaign'}
                    }
                )
            ) }}
        {% endfor %}
        </ul>
    </div>
</div>
<!-- BLOCK: END -->
```