The LandingPage Field Type

This page describes features available only in eZ Studio.

Overview
Landing Page Field Type defines the layout structure of the Landing Page

Name	Internal name	Expected input
`Landing page`	`ezlandingpage`	`string (JSON)`

With the Landing Page Field Type you can define a layout with multiple zones within a single Landing Page.

Within each zone, editors can create blocks, the basic segments of a Landing Page, that contain particular content categories. Specific Content items, called block items, can be added to these blocks.

The configuration of layout and blocks is stored in the **default_layouts.yml** file. For more information about configuration, refer to the EzLandingPageFieldTypeBundle Configuration chapter .

This organization of the layout structure is particularly useful for managing homepages.

In this topic

Layout

Layout is the way in which a Landing Page is divided into zones.

- By default, Studio Demo Bundle provides several preset layouts.

Select a Layout window with different layouts to choose

A Landing Page can have one of two general layout types:

- **Static page** is a type of Landing Page with a very basic content structure. It offers very flexible editing of content and layout at the same time, which is useful for example in simple marketing campaigns.

  The **Static** Landing Page is beyond the scope of the first stable release (eZ Enterprise 2016.)

- **Dynamic page** is a type of Landing Page with an advanced content structure. It introduces the possibility to manipulate the structure of pages. The dynamic structure is based on zones and easily deployable blocks.

- You can use any type of layout from the Studio Demo bundle or you can create a new one from scratch. You can define as many layouts as you need.

Zones

As mentioned above, a layout is composed of zones. Zones are organized structures that are deployed over a layout in a particular position .

The placement of zones is defined in a template which is a part of the layout configuration. You can modify the template, hard-coded in HTML, in order to define your own system of zones.

Zone structure

Each zone contains the following parameters:

Name	Description
<zone_id>	*Required*. A unique zone ID
<name>	*Required*. Zone name

Defining a zone layout

You can define a new layout file (e.g. in Twig) for a zone and include it in a Landing page layout.

A Zone is a container for blocks. The best way to display blocks in the zone is to iterate over a blocks array and render the blocks in a loop.

> For eZ Studio, the **`data-studio-zone`** attribute is required to allow dropping the Content into specific zones.

Example of a zone layout:

zone.html.twig

```
<div data-studio-zone="{{ zones[0].id }}">
 {# If a zone with [0] index contains any blocks #}
 {% if zones[0].blocks %}
     {# for each block #}
  {% for block in zone[0].blocks %}
      {# create a new layer with appropriate id #}
   <div class="landing-page__block block_{{ block.type }}">
          {# render the block by using the "ez_block:renderBlockAction" controller
#}
    {{ render_esi(controller('ez_block:renderBlockAction', {
               {# id of the current content #}
      'contentId': contentInfo.id,
            {# id of the current block #}
      'blockId': block.id
           }))
    }}
      </div>
  {% endfor %}
 {% endif %}
</div>
```

Blocks

Blocks are the basic segment of a Landing Page and integral parts of a zone. Each zone can contain a number of blocks.

Blocks can be placed on a Landing Page using drag-and-drop and later customized.

Creating a new block

Creating a class for the block

The class for the block must implement the **`BlockType`** interface:

```
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\BlockType
```

Most methods are implemented in a universal way by using the **`AbstractBlockType`** abstract class:

```
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\AbstractBlockType
```

If your block does not have specific attributes or a structure, you can extend the **`AbstractBlockType`** class, which contains simple generic converters designated for the block attributes.

Example:

```
<?php
namespace AcmeDemoBundle\Block;

use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\AbstractBlockType;

/**
* RSS block
* Renders feed from a given URL.
*/
class RSSBlock extends AbstractBlockType
{
   // Class body
}
```

Describing a class definition

A block **must** have a definition set using two classes:

The **`BlockAttributeDefinition`** class defines the attributes of a block:

Attribute	Type	Definition
$id	string	block attribute ID
$name	string	block attribute name

The **`BlockDefinition`** class describes a block:

Attribute	Type	Definition	Note
$type	string	block type	
$name	string	block name	
$category	string	block category	

$type	string	block attribute type, available options are: • `integer` • `string` • `url` • `text` • `embed` • `select` • `multiple`
$regex	string	block attribute regex used for validation
$regexErrorMessage	string	message displayed when regex does not match
$required	bool	TRUE if attribute is required
$inline	bool	indicates whether block attribute input should be rendered inline in a form
$values	array	array of chosen values
$options	array	array of available options

$thumbnail	string	path to block thumbnail image	
$templates	array	array of available paths of templates	Retrieved from the config file (default_layouts.yml)
$attributes	array	array of block attributes (objects of **`BlockAttributeDefinition`** class)	

When extending **`AbstractBlockType`** you **must** implement at least 3 methods:

- createBlockDefinition()

  This method must return an **`EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Definition\BlockDefinition`** object.

  Example of a Gallery block:

```
/**
 * Creates BlockDefinition object for block type.
 *
 * @return
\EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Definition\BlockDefin
ition
 */
public function createBlockDefinition()
{
    return new BlockDefinition(
        'gallery',
        'Gallery Block',
        'default',
        'bundles/ezsystemslandingpagefieldtype/images/thumbnails/gallery.svg',
        [],
        [
            new BlockAttributeDefinition(
                'contentId',
                'Folder',
                'embed',
                '/^([a-zA-Z]:)?(\/[a-zA-Z0-9_\/-]+)+\/?/',
                'Choose an image folder'
            ),
        ]
    );
}
```

getTemplateParameters(BlockValue $blockValue)

This method returns an array of parameters to be displayed in rendered view of block. You can access them directly in a block template (e. g. via twig `{{ title }}`).

When parameters are used in the template you call them directly without the `parameters` array name:

Correct	Not Correct
`<h1>{{ title }}</h1>`	~~`<h1>{{ parameters.title }}</h1>`~~

Example of the `getTemplateParameters()` method implementation:

```
/**
* @param
\EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\BlockValue
$blockValue
*
* @return array
*/
public function getTemplateParameters(BlockValue $blockValue)
{
 $attributes = $blockValue->getAttributes();
 $limit = (isset($attributes['limit'])) ? $attributes['limit'] : 10;
 $offset = (isset($attributes['offset'])) ? $attributes['offset'] : 0;
 $parameters = [
  'title' => $attributes['title'],
  'limit' => $limit,
  'offset' => $offset,
  'feeds' => $this->RssProvider->getFeeds($attributes['url']),
 ];

 return $parameters;
}
```

checkAttributesStructure(array $attributes)

This method validates the input fields for a block. You can specify your own conditions to throw the `InvalidBlockAttributeExcepti` on exception.

This `InvalidBlockAttributeException` exception has the following parameters:

Name	Description
blockType	name of a block
attribute	name of the block's attribute which failed validation
message	a short information about an error
previous	previous exception, null by default

Example:

```
/**
 * Checks if block's attributes are valid.
 *
 * @param array $attributes
 *
 * @throws
\EzSystems\LandingPageFieldTypeBundle\Exception\InvalidBlockAttributeException
 */
public function checkAttributesStructure(array $attributes)
{
    if (!isset($attributes['url'])) {
        throw new InvalidBlockAttributeException('RSS', 'url', 'URL must be
set.');
    }

    if (isset($attributes['limit']) && (($attributes['limit'] < 1) ||
(!is_numeric($attributes['limit'])))) {
        throw new InvalidBlockAttributeException('RSS', 'limit', 'Limit must be a
number greater than 0.');
    }

    if (isset($attributes['offset']) && (($attributes['offset'] < 0) ||
(!is_numeric($attributes['limit'])))) {
        throw new InvalidBlockAttributeException('RSS', 'offset', 'Offset must be
a number no less than 0.');
    }
}
```

When the class is created make sure it is added to a container.

Adding the class to the container

The **services.yml** file must contain info about your block class.

The description of your class must contain a tag which provides:

- tag name: **landing_page_field_type.block_type**
- tag alias: <name of a block>

An example:

```
acme.landing_page.block.rss:            # service id
       class: AcmeDemoBundle\FieldType\LandingPage\Model\Block\RSSBlock # block's
class with namespace
       tags:                # service definition must contain tag with
           - { name: landing_page_field_type.block_type, alias: rss} #
"landing_page_field_type.block_type" name and block name as an alias
```

Rendering Landing Page

Landing page rendering takes place while editing or viewing.

When rendering a Landing Page, its zones are passed to the layout as a `zones` array with a `blocks` array each. You can simply access them using twig (e.g. **`{{ zones[0].id }}`**).

> Each div that's a zone or zone's container should have data attributes:
>
> - **`data-studio-zones-container`** for a div containing zones
> - **`data-studio-zone`** with zone ID as a value for a zone container

To render a block inside the layout, use twig ***`render_esi()`*** function to call **`ez_block:renderBlockAction`** .

> **`ez_block`** is an alias to **`EzSystems\LandingPageFieldTypeBundle\Controller\BlockController`**

An action has the following parameters:

- **`contentId`** - ID of content which can be accessed by **`contentInfo.id`**
- **`blockId`** - ID of block which you want to render

Example usage:

```
{{ render_esi(controller('ez_block:renderBlockAction', {
  'contentId': contentInfo.id,
     'blockId': block.id
 }))
}}
```

As a whole a sample layout could look as follows:

landing_page_simple_layout.html.twig

```
{# a layer of the required "data-studio-zones-container" attribute, in which zones
will be displayed #}
<div data-studio-zones-container>
  {# a layer of the required attribute for the displayed zone #}
  <div data-studio-zone="{{ zones[0].id }}">
  {# If a zone with [0] index contains any blocks #}
  {% if zones[0].blocks %}
      {# for each block #}
   {% for block in blocks %}
       {# create a new layer with appropriate id #}
    <div class="landing-page__block block_{{ block.type }}">
          {# render the block by using the "ez_block:renderBlockAction" controller
#}
     {{ render_esi(controller('ez_block:renderBlockAction', {
                {# id of the current content #}
       'contentId': contentInfo.id,
                {# id of the current block #}
       'blockId': block.id
            }))
     }}
       </div>
   {% endfor %}
  {% endif %}
    </div>
</div>
```

Viewing template

Your view is populated with data (parameters) retrieved from the **`getTemplateParameters()`** method which must be implemented in your block's class.

Example:

```
/**
    * @param
\EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\BlockValue
$blockValue
    *
    * @return array
    */
   public function getTemplateParameters(BlockValue $blockValue)
   {
       $attributes = $blockValue->getAttributes();
       $limit = (isset($attributes['limit'])) ? $attributes['limit'] : 10;
       $offset = (isset($attributes['offset'])) ? $attributes['offset'] : 0;
       $parameters = [
           'title' => $attributes['title'],
           'limit' => $limit,
           'offset' => $offset,
           'feeds' => $this->RssProvider->getFeeds($attributes['url']),
       ];
       return $parameters;
   }
```

Implementation Example

Block Class

TagBlock.php

```
<?php
/**
 * @copyright Copyright (C) eZ Systems AS. All rights reserved.
 * @license For full copyright and license information view LICENSE file distributed
with this source code.
 */
namespace EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\Block;

use EzSystems\LandingPageFieldTypeBundle\Exception\InvalidBlockAttributeException;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Definition\BlockDefinitio
n;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Definition\BlockAttribute
Definition;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\AbstractBlockType;
use EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\BlockType;
use EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\BlockValue;

/**
 * Tag block
 * Renders simple HTML.
 */
class TagBlock extends AbstractBlockType implements BlockType
{
```

```
    /**
     * Returns array of parameters required to render block template.
     *
     * @param array $blockValue Block value attributes
     *
     * @return array Template parameters
     */
    public function getTemplateParameters(BlockValue $blockValue)
    {
        return ['block' => $blockValue];
    }

 /**
   * Creates BlockDefinition object for block type.
   *
   * @return
\EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Definition\BlockDefiniti
on
   */
    public function createBlockDefinition()
    {
        return new BlockDefinition(
            'tag',
            'Tag Block',
            'default',
            'bundles/ezsystemslandingpagefieldtype/images/thumbnails/tag.svg',
            [],
            [
                new BlockAttributeDefinition(
                    'content',
                    'Content',
                    'text',
                    '/[^\\s]/',
                    'Provide html code'
                ),
            ]
        );
    }

 /**
   * Checks if block's attributes are valid.
  *
  * @param array $attributes
  *
  * @throws
\EzSystems\LandingPageFieldTypeBundle\Exception\InvalidBlockAttributeException
  */
    public function checkAttributesStructure(array $attributes)
    {
        if (!isset($attributes['content'])) {
            throw new InvalidBlockAttributeException('Tag', 'content', 'Content must
be set.');
        }
    }
```

```
}
```

service.yml configuration

services.yml

```
ezpublish.landing_page.block.tag:
    class:
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPage\Model\Block\TagBlock
    tags:
        - { name: landing_page_field_type.block_type, alias: tag }
```

Block template

tag.html.twig

```
{{ block.attributes.content|raw }}
```