Creating Landing Page blocks (Enterprise)

Description

V1.2

A Landing Page has a customizable layout with multiple zones where you can place predefined blocks with content.

By default eZ Enterprise comes with a number of preset Landing Page blocks. You can, however, add custom blocks to your configuration.

Solution

Block configuration

In the Demo installation the layout configuration is stored in `ezstudio-demo-bundle/Resources/config/default_layouts.yml`:

Example default_layouts.yml

```
blocks:
    gallery:
        views:
            gallery:
                template:
eZStudioDemoBundle:blocks:gallery.html.twig
                name: Default Gallery Block template
    keyword:
        views:
            keyword:
                template:
eZStudioDemoBundle:blocks:keyword.html.twig
                name: Default Keyword Block template
    rss:
        views:
            rss:
                template:
eZStudioDemoBundle:blocks:rss.html.twig
                name: Default RSS Block template
    tag:
        views:
            tag:
                template:
eZStudioDemoBundle:blocks:tag.html.twig
                name: Default Tag Block template
```

Creating a new block

Creating a class for the block

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

In this topic:

Related topics:

Creating Landing Page layouts (Enterprise)

Landing Page Field Type (Enterprise)

```
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\Model\BlockType
```

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

```
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\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.

For example:

```
<?php
namespace AcmeDemoBundle\Block;

use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\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:

BlockAttributeDefinition

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

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

BlockDefinition

The `BlockDefinition` class describes a block:

Attribute	Type	Definition	Note
$type	string	block type	
$name	string	block name	
$category	string	block category	
$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\Landi
ngPage\Definition\BlockDefinition
 */
public function createBlockDefinition()
{
    return new BlockDefinition(
        'gallery',
        'Gallery Block',
        'default',

'bundles/ezsystemslandingpagefieldtype/images/thumbna
ils/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\Landi
ngPage\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 `InvalidBlockAttributeException` 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

For example:

```
/**
 * Checks if block's attributes are valid.
 *
 * @param array $attributes
 *
 * @throws
\EzSystems\LandingPageFieldTypeBundle\Exception\Inval
idBlockAttributeException
 */
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>

For example:

```
acme.landing_page.block.rss:            # service id
       class:
AcmeDemoBundle\FieldType\LandingPage\Model\Block\RSSBloc
k # 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
```

Custom editing UI

If you want to add a custom editing UI to your new block, you need to provide the code for the custom popup UI in Javascript (see the code for eZS.ScheduleBlockView or eZS.TagBlockView for examples).

Once it is ready, create a plugin for `eZS.LandingPageCreatorView` that makes a use of the `addBlock` public method from `eZS.LandingPageCreatorView`, see the example below:

```
YUI.add('ezs-addcustomblockplugin', function (Y) {
    'use strict';

    var namespace = 'Any.Namespace.Of.Your.Choice',

    Y.namespace(namespace);
    NS = Y[namespace];

    NS.Plugin.AddCustomBlock =
Y.Base.create('addCustomBlockPlugin', Y.Plugin.Base, [],
{
        initializer: function () {
            this.get('host').addBlock('custom',
NS.CustomBlockView);
        },
    }, {
        NS: 'dashboardPlugin'
    });

    Y.eZ.PluginRegistry.registerPlugin(
        NS.Plugin.AddCustomBlock,
['landingPageCreatorView']
    );
});
```

Upcoming feature - multiple block templates
The ability to configure different templates (views) for one Landing Page block is upcoming. See EZS-1008 to follow its progress.

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\LandingPa
ge\Model\Block;

use
EzSystems\LandingPageFieldTypeBundle\Exception\InvalidBl
ockAttributeException;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\Definition\BlockDefinition;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\Definition\BlockAttributeDefinition;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\Model\AbstractBlockType;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\Model\BlockType;
use
EzSystems\LandingPageFieldTypeBundle\FieldType\LandingPa
ge\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\LandingP
age\Definition\BlockDefinition
   */
    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\InvalidB
lockAttributeException
  */
    public function checkAttributesStructure(array
$attributes)
    {
        if (!isset($attributes['content'])) {
            throw new
InvalidBlockAttributeException('Tag', 'content',
'Content must be set.');
        }
    }
```

```
}
```

V1.7

If you want to make sure that your block is only available in the Element menu in a specific situation, you can override the `isAvailable` method, which makes the block accessible by default:

```
public function isAvailable()
    {
        return true;
    }
```

service.yml configuration

services.yml

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

Block template

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