How to use a custom controller to display a content item or location

Enhanced views for Content/Location

In some cases, displaying a Content item/Location via the built-in `ViewController` is not sufficient and will force you to make many sub-requests in order to access different parameters.

Typical use cases are access to:

- Settings (either coming from `ConfigResolver` or `ServiceContainer`)
- Current Content item's `ContentType` object
- Current Location's parent
- Current Location's children count
- Main Location and alternative Locations for the current Content item
- etc.

In those cases, you may want to **use your own controller** to display the current Content item/Location instead of using the built-in `ViewController`.

Description

This feature covers 2 general use cases:

- Lets you configure a custom controller with the configured matcher rules.
- Lets you override the built-in view controller in a clean way.

Matching custom controllers

This is possible with the following piece of configuration:

```
ezpublish:
    system:
        my_siteaccess:
            location_view:
                full:
                    # Defining a ruleset matching a location and pointing to a
controller
                    my_ruleset:
                        # The following will let you use your own custom controller
for location #123
                        # (Here it will use
AcmeTestBundle/Controller/DefaultController::viewLocationAction(),
                        # following the Symfony controller notation convention.
                        # Method viewLocationAction() must follow the same prototype
as in the built-in ViewController
                        controller: AcmeTestBundle:Default:viewLocation
                        match:
                            Id\Location: 123
```

> You can point to any kind of controller supported by Symfony (including controllers as a service).

The only requirement here is that your action method has a similar signature than `ViewController::viewLocation()` or `ViewController::viewContent()` (depending on what you're matching of course). However, note that all arguments are not mandatory since Symfony is clever enough to know what to inject into your action method . That is why **you aren't forced to mimic the `ViewController`'s signature strictly**. For example, if you omit `$layout` and `$params` arguments, it will still be valid. Symfony will just avoid injecting them into your action method.

Original ViewController signatures

viewLocation() signature

```
/**
 * Main action for viewing content through a location in the repository.
 *
 * @param int $locationId
 * @param string $viewType
 * @param boolean $layout
 * @param array $params
 *
 * @throws \Symfony\Component\Security\Core\Exception\AccessDeniedException
 * @throws \Exception
 *
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function viewLocation( $locationId, $viewType, $layout = false, array $params =
array() )
```

viewContent() signature

```
/**
 * Main action for viewing content.
 *
 * @param int $contentId
 * @param string $viewType
 * @param boolean $layout
 * @param array $params
 *
 * @throws \Symfony\Component\Security\Core\Exception\AccessDeniedException
 * @throws \Exception
 *
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function viewContent( $contentId, $viewType, $layout = false, array $params =
array() )
```

Note
Controller selection doesn't apply to `block_view` since you can already use your own controller to display blocks.

Warning on caching
Using your own controller, **it is your responsibility to define cache rules**, like for every custom controller !

So don't forget to **set cache rules** and the appropriate **`X-Location-Id` header** in the returned `Response` object.

See built-in ViewController for more details on this.

Examples

Enriching built-in ViewController

This example shows how to use a custom controller to enrich the final configured view template. Your controller will here forward the request to the built-in `ViewController` with some additional parameters.

This is usually the recommended way to use a custom controller.

Always ensure that you add new parameters to existing `$params` associative array, using **+** union operator or `array_merge()` .
Not doing so (e.g. only passing your custom parameters array) can result in unexpected issues with content preview.
Previewed content and other parameters are indeed passed in `$params`.

ezplatform.yml

```
ezpublish:
    system:
        ezdemo_frontend_group:
            location_view:
                full:
                    article_test:
                        # Configuring both controller and template as the controller
will forward
                        # the request to the ViewController which will render the
configured template.
                        controller: AcmeTestBundle:Default:articleViewEnhanced
                        template: AcmeTestBundle:full:article_test.html.twig
                        match:
                            Identifier\ContentType: [article]
```

Controller

```
<?php
namespace Acme\TestBundle\Controller;
use Symfony\Component\HttpFoundation\Response;
use eZ\Bundle\EzPublishCoreBundle\Controller;

class DefaultController extends Controller
{
    public function articleViewEnhancedAction( $locationId, $viewType, $layout =
false, array $params = array() )
    {
        // Add custom parameters to existing ones.
        $params += array( 'myCustomVariable' => "Hey, I'm a custom message!" );
        // Forward the request to the original ViewController
        // And get the response. Eventually alter it (here we change the smax-age for
cache).
        $response = $this->get( 'ez_content' )->viewLocation( $locationId, $viewType,
$layout, $params );
        $response->setSharedMaxAge( 600 );

        return $response;
    }
}
```

article_test.html.twig

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

{% block content %}
    <h1>{{ ez_render_field( content, 'title' ) }}</h1>
    <h2>{{ myCustomVariable }}</h2>
    {{ ez_render_field( content, 'body' ) }}
{% endblock %}
```

Using a custom controller to get full control

This example shows you how to configure and use your own controller to handle a location.

ezplatform.yml

```
ezpublish:
    system:
        ezdemo_frontend_group:
            location_view:
                full:
                    my_ruleset:
                        controller: AcmeTestBundle:Default:viewFolder
                        match:
                            Identifier\ContentType: [folder]
                            Identifier\Section: [standard]
```

Always ensure to have a `$params` argument and add new parameters to it, using **+** union operator or `array_merge()` .

Not doing so (e.g. only passing your custom parameters array) can result in unexpected issues with content preview.
Previewed content and other parameters are indeed passed in `$params`.

Controller

```
<?php
namespace Acme\TestBundle\Controller;
use Symfony\Component\HttpFoundation\Response;
use eZ\Bundle\EzPublishCoreBundle\Controller;

class DefaultController extends Controller
{
    public function viewFolderAction( $locationId, $layout = false, $params = array()
)
    {
        $repository = $this->getRepository();
        $location = $repository->getLocationService()->loadLocation( $locationId );
        // Check if content is not already passed. Can be the case when using content
preview.
        $content = isset( $params['content'] ) ? $params['content'] :
$repository->getContentService()->loadContentByContentInfo(
$location->getContentInfo() )
        $response = new Response();
        $response->headers->set( 'X-Location-Id', $locationId );
        // Caching for 1h and make the cache vary on user hash
        $response->setSharedMaxAge( 3600 );
        $response->setVary( 'X-User-Hash' );
        return $this->render(
            'AcmeTestBundle::custom_controller_folder.html.twig',
            array(
                'location' => $location,
                'content' => $content,
                'foo' => 'Hey world!!!',
                'osTypes' => array( 'osx', 'linux', 'losedows' )
            ) + $params
        );
    }
}
```

custom_controller_folder.html.twig

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

{% block content %}
<h1>{{ ez_render_field( content, 'title' ) }}</h1>
    <h1>{{ foo }}</h1>
    <ul>
    {% for os in osTypes %}
        <li>{{ os }}</li>
    {% endfor %}
    </ul>
{% endblock %}
```

Overriding the built-in ViewController

One other way to keep control of what is passed to the view is to use your own controller instead of the built-in ViewController.

Base ViewController being defined as a service, with a service alias, this can be easily achieved from your bundle's configuration:

```
parameters:
    my.custom.view_controller.class: Acme\TestBundle\MyViewController

services:
    my.custom.view_controller:
        class: %my.custom.view_controller.class%
        arguments: [@some_dependency, @other_dependency]

    # Change the alias here and make it point to your own controller
    ez_content:
        alias: my.custom.view_controller
```

Warning
Doing so will completely override the built-in ViewController! Use this at your own risk!

See also
See also

How to Display a default text while asynchronous loading of a controller

How to render an embedded content from a Twig template