View provider configuration

The **configured ViewProvider** allows to configure template selection when using the `ViewController`, either directly form a URL or via a sub-request.

> **eZ Publish 4.x terminology**
> In eZ Publish 4.x, it was known as **template override system *by configuration*** (`override.ini`).
> However this only reflects old overrides for `node/view/*.tpl` and `content/view/*.tpl`.

Principle

The **configured ViewProvider** takes its configuration from your siteaccess in the **`location_view/content_view`** section (most of the time you want to match a location, so we'll focus on `location_view` configuration). This configuration is a hash built in the following way:

```
# ezpublish/config/ezpublish.yml
ezpublish:
    system:
        # Can be a valid siteaccess, siteaccess group or even "global"
        front_siteaccess:
            # Configuring the LocationViewProvider
            location_view:
                # The view type (full/line are standard, but you can use custom ones)
                full:
                    # A simple unique key for your matching ruleset
                    folderRuleset:
                        # The template identifier to load, following the Symfony
bundle notation for templates
                        # See
http://symfony.com/doc/current/book/controller.html#rendering-templates
                        template: eZDemoBundle:full:small_folder.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.
                            Identifier\ContentType: [small_folder, folder]
```

> **Important note about template matching**
> **Template matching will NOT work if your content contains a field type that is not supported by the repository**. It can be the case when you are in the process of a migration from eZ Publish 4.x, where custom datatypes have been developed.
> In this case the repository will throw an exception which is caught in the `ViewController`, **causing the fallback to the legacy kernel** .
>
> The list of field types supported out of the box is available here.

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

Matchers

To be able to select the right templates against conditions, the view provider uses matcher objects, all implementing `eZ\Publish\Core\MVC\Symfony\View\ContentViewProvider\Configured\Matcher` interface.

Matcher identifier

The matcher identifier can comply to 3 different formats:

1. **Relative qualified class name** (e.g. `Identifier\ContentType`). This is the most common case and used for native matchers. It will then be relative to `eZ\Publish\Core\MVC\Symfony\View\ContentViewProvider\Configured\Matcher`.
2. **Full qualified class name** (e.g. `\Foo\Bar\MyMatcher`). This is a way to specify a **custom matcher** that doesn't need specific dependency injection. Please note that it **must** start with a `\`.
3. **Service identifier**, as defined in Symfony service container. This is the way to specify a more **complex custom matcher** that has dependencies.

> **Injecting the Repository**
> If your matcher needs the repository, simply make it implement `eZ\Publish\Core\MVC\RepositoryAwareInterface` or simply extend `eZ\Publish\Core\MVC\RepositoryAware` abstract class. The repository will then be correctly injected before matching.

Matcher value

The value associated to the matcher is being passed to its `setMatchingConfig()` method and can be anything supported by the matcher.

> In the case of native matchers, they support both **scalar values** or **arrays of scalar values**.
> Passing an array amounts to applying a logical OR.

Combining matchers

It is possible to combine matchers to add additional constraints for matching a template:

```
# ...
match:
    Identifier\ContentType: [small_folder, folder]
    Identifier\ParentContentType: frontpage
```

The example above results to say "Match any content which **ContentType** identifier is ***small_folder* OR *folder***, **AND** having *frontpage* as **Parent ContentType** identifier".

Available matchers

The following table presents all native matchers.

Identifier	Description
`Id\Content`	Matches the ID number of the content
`Id\ContentType`	Matches the ID number of the content type whose content is an instance of
`Id\ContentTypeGroup`	Matches the ID number of the group of the content type whose content is an instance of belongs to
`Id\Location`	Matches the ID number of a location. *In the case of a Content, matched against the main location.*
`Id\ParentContentType`	Matches the ID number of the parent content type. *In the case of a Content, matched against the main location.*
`Id\ParentLocation`	Matches the ID number of the parent location. *In the case of a Content, matched against the main location.*

`Id\Remote`	Matches the remoteId of either content or location, depending on the object matched.
`Id\Section`	Matches the ID number of the section whose content belongs to
`Id\State`	*Not supported yet.*
`Identifier\ContentType`	Matches the identifier of the content type whose content is an instance of
`Identifier\ParentContentType`	Matches the identifier of the parent content type. *In the case of a Content, matched against the main location.*
`Identifier\Section`	Matches the identifier of the section whose content belongs to
`Identifier\State`	*Not supported yet.*
`Depth`	Matches the depth of the location. The depth of a top level location is 1.
`UrlAlias`	Matches the virtual URL of the location (i.e. `/My/Content-Uri`). **Important: Matches when the UrlAlias of the location starts with the value passed.** *Not supported for Content (aka content_view).*