Dynamic settings injection

Description

Before 5.4, if you wanted to implement a service needing siteaccess-aware settings (e.g. language settings), you needed to inject the whole `ConfigResolver` (`ezpublish.config.resolver`) and get the needed settings from it. This was neither very convenient nor explicit.

The goal of this feature is to allow developers to inject these dynamic settings explicitly from their service definition (yml, xml, annotation, etc.).

Usage

Syntax

Static container parameters follow the `%<parameter_name>%` syntax in Symfony.

Dynamic parameters have the following: `$<parameter_name>[; <namespace>[; <scope>]]$`, default namespace being `ezsettings`, and default scope being the current siteaccess.

> For more information, see ConfigResolver documentation.

DynamicSettingParser

This feature also introduces a *DynamicSettingParser* service that can be used for adding support of the dynamic settings syntax.
This service has `ezpublish.config.dynamic_setting.parser` for ID and implements `eZ\Bundle\EzPublishCoreBundle\DependencyInjection\Configuration\SiteAccessAware\DynamicSettingParserInterface`.

Limitations

A few limitations still remain:

- It is not possible to use dynamic settings in your semantic configuration (e.g. `config.yml` or `ezplatform.yml`) as they are meant primarily for parameter injection in services.
- It is not possible to define an array of options having dynamic settings. They will not be parsed. Workaround is to use separate arguments/setters.
- Injecting dynamic settings in request listeners is **not recommended**, as it won't be resolved with the correct scope (request listeners are **instantiated before SiteAccess match**). Workaround is to inject the ConfigResolver instead, and resolving the setting in your `onKernelRequest` method (or equivalent).

Examples

Injecting an eZ parameter

Defining a simple service needing `languages` parameter (i.e. prioritized languages).

Before 5.4

```
parameters:
    acme_test.my_service.class: Acme\TestBundle\MyServiceClass

services:
    acme_test.my_service:
        class: %acme_test.my_service.class%
        arguments: [@ezpublish.config.resolver]

namespace Acme\TestBundle;
```

```
use eZ\Publish\Core\MVC\ConfigResolverInterface;

class MyServiceClass
{
    /**
 * Prioritized languages
 *
 * @var array
 */
    private $languages;

    public function __construct( ConfigResolverInterface $configResolver )
    {
        $this->languages = $configResolver->getParameter( 'languages' );
    }
}
```

After, with setter injection (preferred)

```
parameters:
    acme_test.my_service.class: Acme\TestBundle\MyServiceClass

services:
    acme_test.my_service:
        class: %acme_test.my_service.class%
        calls:
            - [setLanguages, ["$languages$"]]
```

```
namespace Acme\TestBundle;

class MyServiceClass
{
    /**
     * Prioritized languages
     *
     * @var array
     */
    private $languages;

    public function setLanguages( array $languages = null )
    {
        $this->languages = $languages;
    }
}
```

Important: Ensure you always add `null` as a default value, especially if the argument is type-hinted.

After, with constructor injection

```
parameters:
    acme_test.my_service.class: Acme\TestBundle\MyServiceClass

services:
    acme_test.my_service:
        class: %acme_test.my_service.class%
        arguments: ["$languages$"]
```

```
namespace Acme\TestBundle;

class MyServiceClass
{
    /**
     * Prioritized languages
     *
     * @var array
     */
    private $languages;

    public function __construct( array $languages )
    {
        $this->languages = $languages;
    }
}
```

Injecting 3rd party parameters

```
parameters:
    acme_test.my_service.class: Acme\TestBundle\MyServiceClass
    # "acme" is our parameter namespace.
    # Null is the default value.
    acme.default.some_parameter: ~
    acme.ezdemo_site.some_parameter: foo
    acme.ezdemo_site_admin.some_parameter: bar

services:
    acme_test.my_service:
        class: %acme_test.my_service.class%
        # The following argument will automatically resolve to the right value,
depending on the current SiteAccess.
        # We specify "acme" as the namespace we want to use for parameter resolving.
        calls:
            - [setSomeParameter, ["$some_parameter;acme$"]]
```

```
namespace Acme\TestBundle;
class MyServiceClass
{
    private $myParameter;
    public function setSomeParameter( $myParameter = null )
    {
        // Will be "foo" for ezdemo_site, "bar" for ezdemo_site_admin, or null if
another SiteAccess.
        $this->myParameter = $myParameter;
    }
}
```