

# Siteaccess Matching

- [Configuration](#)
- [Available matchers](#)
- [Compound siteaccess matcher](#)
- [Matching by request header](#)
- [Matching by environment variable](#)
- [URILexer and semanticPathinfo](#)

Siteaccess matching is done through **eZ\Publish\MVC\SiteAccess\Matcher** objects. You can configure this matching and even develop custom matchers.

## Configuration

You can configure siteaccess matching in your main **ezpublish/config/ezpublish.yml** :

### ezpublish.yml

```
ezpublish:
  siteaccess:
    default_siteaccess: ezdemo_site
    list:
      - ezdemo_site
      - eng
      - fre
      - fr_eng
      - ezdemo_site_admin
    groups:
      ezdemo_site_group:
        - ezdemo_site
        - eng
        - fre
        - fr_eng
        - ezdemo_site_admin
    match:
      Map\URI:
        ezdemo_site: ezdemo_site
        fre: fre
        ezdemo_site_admin: ezdemo_site_admin
```

You need to set several parameters:

- **ezpublish.siteaccess.default\_siteaccess**
- **ezpublish.siteaccess.list**
- *(optional)* **ezpublish.siteaccess.groups**
- **ezpublish.siteaccess.match**

**ezpublish.siteaccess.default\_siteaccess** is the default siteaccess that will be used if matching was not successful. This ensures that a siteaccess is always defined.

**ezpublish.siteaccess.list** is the list of all available siteaccesses in your website.


*(optional)* **ezpublish.siteaccess.groups** defines which groups siteaccesses are member of. This is useful when you want to mutualize settings between several siteaccesses and avoid config duplication. Siteaccess groups are considered as regular siteaccesses as far as configuration is concerned.



A siteaccess can be part of several groups.


A siteaccess configuration has always precedence on the group configuration.

**ezpublish.siteaccess.match** holds the matching configuration. It consists in a hash where the key is the name of the matcher class. If the matcher class doesn't start with a `\`, it will be considered relative to `eZ\Publish\MVC\SiteAccess\Matcher` (e.g. `Map\Host` will refer to `eZ\Publish\MVC\SiteAccess\Matcher\Map\Host`)



Every **custom matcher** can be specified with a **fully qualified class name** (e.g. `\My\SiteAccess\Matcher`) or by a **service identifier prefixed by @** (e.g. `@my_matcher_service`).

- In the case of a fully qualified class name, the matching configuration will be passed in the constructor.
- In the case of a service, it must implement `eZ\Bundle\EzPublishCoreBundle\SiteAccess\Matcher`. The matching configuration will be passed to `setMatchingConfiguration()`.



Make sure to type matcher in correct case, if wrong case like "Uri" instead of "URI" it will happily work on systems like Mac OS X because of case in sensitive file system, while it will fail when you deploy it to a linux server. This is a known artifact of [PSR-0 autoloading](#) of PHP classes.

Available matchers

Name	Description	Configuration	Example
URIElement	Maps a URI element to a siteaccess. This is the default matcher used when choosing URI matching in setup wizard.	<div>The element number you want to match (starting from 1).</div> <div><pre>ezpublish: siteaccess:  match:  URIElement: 1</pre></div> <div><b>Important:</b> When using a value &gt; 1, it will concatenate the elements with <code>_</code></div>	<p><b>URI:</b> <code>/ezdemo_site/foo/bar</code></p> <p><u>Element number:</u> 1 <u>Matched siteaccess:</u> <code>ezdemo_site</code></p> <p><u>Element number:</u> 2 <u>Matched siteaccess:</u> <code>ezdemo_site_foo</code></p>
URIText	Matches URI using <i>pre</i> and/or <i>post</i> sub-strings in the first URI segment	<div>The prefix and/or suffix (none are required)</div> <div><pre>ezpublish: siteaccess:  match:  URIText:  prefix: foo  suffix: bar</pre></div>	<p><b>URI:</b> <code>/footestbar/my/content</code></p> <p><u>Prefix:</u> <code>foo</code> <u>Suffix:</u> <code>bar</code> <u>Matched siteaccess:</u> <code>test</code></p>

HostElement	Maps an element in the host name to a siteaccess.	<p>The element number you want to match (starting from 1).</p> <pre> ezpublish:  siteaccess:  match:  HostElement: 2 </pre>	<p><b>Host name:</b> <code>www.example.com</code></p> <p><u>Element number:</u> 2</p> <p><u>Matched siteaccess:</u> example</p>
HostText	Matches a siteaccess in the host name, using <i>pre</i> and/or <i>post</i> sub-strings.	<p>The prefix and/or suffix (none are required)</p> <pre> ezpublish:  siteaccess:  match:  HostText:  prefix: www.  suffix: .com </pre>	<p><b>Host name:</b> <code>www.foo.com</code></p> <p><u>Prefix:</u> <code>www.</code></p> <p><u>Suffix:</u> <code>.com</code></p> <p><u>Matched siteaccess:</u> foo</p>
Map\Host	Maps a host name to a siteaccess.	<p>A hash map of host/siteaccess</p> <pre> ezpublish:  siteaccess:  match:  Map\Host:  www.foo.com: foo_front  adm.foo.com: foo_admin  www.bar-stuff .fr: bar_front  adm.bar-stuff .fr: bar_admin </pre>	<p><b>Map:</b></p> <ul style="list-style-type: none"> <li>• <code>www.foo.com =&gt; foo_front</code></li> <li>• <code>admin.foo.com =&gt; foo_admin</code></li> </ul> <p><b>Host name:</b> <code>www.example.com</code></p> <p><u>Matched siteaccess:</u> <code>foo_front</code></p>

Map\URI	Maps a URI to a siteaccess	A hash map of URI/siteaccess <div> <pre> ezpublish:  siteaccess:  match:  Map\URI:  something: ezdemo_site  foobar: ezdemo_site_admin </pre> </div>	<b>URI:</b> /something/my/content <b>Map:</b> <ul style="list-style-type: none"> <li>something =&gt; ezdemo_site</li> <li>foobar =&gt; ezdemo_site_admin</li> </ul> <b>Matched siteaccess:</b> ezdemo_site
Map\Port	Maps a port to a siteaccess	A has map of Port/siteaccess <div> <pre> ezpublish:  siteaccess:  match:  Match\Port:  80: foo  8080: bar </pre> </div>	<b>URL:</b> http://ezpublish.dev:8080/my/content <b>Map:</b> <ul style="list-style-type: none"> <li>80: foo</li> <li>8080: bar</li> </ul> <b>Matched siteaccess:</b> bar
Regex\Host	Matches against a regexp and extract a portion of it	The regexp to match against and the captured element to use <div> <pre> ezpublish:  siteaccess:  match:  Regex\Host:  regex: "^(\\w+_sa)\$"  # Default is 1  itemNumber: 1 </pre> </div>	<b>Host name:</b> example_sa <b>regex:</b> ^(\w+)_sa\$ <b>itemNumber:</b> 1 <b>Matched siteaccess:</b> example

Regex\URI	Matches against a regexp and extract a portion of it	The regexp to match against and the captured element to use <div> <pre> ezpublish:  siteaccess:  match:  Regex\URI:  regex: "^/foo(\\w+)bar"  # Default is 1  itemNumber: 1 </pre> </div>	<b>URI:</b> /footestbar/somethin g <b>regex:</b> ^/foo(\\w+)bar <b>itemNumber:</b> 1 <b>Matched siteaccess:</b> test
-----------	--	---	--

## Compound siteaccess matcher

The Compound siteaccess matcher allows to combine several matchers together:

- <http://example.com/en> matches site\_en (match on host=example.com *and* URIElement(1)=en)
- <http://example.com/fr> matches site\_fr (match on host=example.com *and* URIElement(1)=fr)
- <http://admin.example.com> matches site\_admin (match on host=admin.example.com)

Compound matchers cover the legacy **host\_uri** matching feature.

They are based on logical combinations, or/and, using logical compound matchers:

- Compound\LogicalAnd
- Compound\LogicalOr

Each compound matcher will specify two or more sub-matchers. A rule will match if all the matchers, combined with the logical matcher, are positive. The example above would have used Map\Host and Map\Uri., combined with a LogicalAnd. When both the URI and host match, the siteaccess configured with "match" is used.

### ezpublish.yml

```
ezpublish:
  siteaccess:
    match:
      Compound\LogicalAnd:
        # Nested matchers, with their configuration.
        # No need to precise their matching values (true will suffice).
        site_en:
          matchers:
            Map\URI:
              en: true
            Map\Host:
              example.com: true
          match: site_en
        site_fr:
          matchers:
            Map\URI:
              en: true
            Map\Host:
              example.com: true
          match: site_en
      Map\Host:
        admin.example.com: site_admin
```

## Matching by request header

It is possible to define which siteaccess to use by setting a **X-Siteaccess** header in your request. This can be useful for REST requests.

In such case, **X-Siteaccess** must be the **siteaccess name** (e.g. *ezdemo\_site*).

## Matching by environment variable

It is also possible to define which siteaccess to use directly via an **EZPUBLISH\_SITEACCESS** environment variable.

This is recommended if you want to get **performance gain** since no matching logic is done in this case.

You can define this environment variable directly from your web server configuration:

### Apache VirtualHost example

```
# This configuration assumes that mod_env is activated
<VirtualHost *:80>
  DocumentRoot "/path/to/ezpublish5/web/folder"
  ServerName example.com
  ServerAlias www.example.com
  SetEnv EZPUBLISH_SITEACCESS ezdemo_site
</VirtualHost>
```



This can also be done via PHP-FPM configuration file, if you use it. See [PHP-FPM documentation](#) for more information.



#### Note about precedence

The precedence order for siteaccess matching is the following (the first matched wins):

1. Request header
2. Environment variable
3. Configured matchers

## URILexer and semanticPathinfo

In some cases, after matching a siteaccess, it is necessary to modify the original request URI. This is for example needed with URI-based matchers since the siteaccess is contained in the original URI and it is not part of the route itself.

The problem is addressed by *analyzing* this URI and by modifying it when needed through the **URILexer** interface.

### URILexer interface

```
/**
 * Interface for SiteAccess matchers that need to alter the URI after matching.
 * This is useful when you have the siteaccess in the URI like
 */
"/<siteaccessName>/my/awesome/uri"
*/
interface URILexer
{
    /**
     * Analyses $uri and removes the siteaccess part, if needed.
     *
     * @param string $uri The original URI
     * @return string The modified URI
     */
    public function analyseURI( $uri );
    /**
     * Analyses $linkUri when generating a link to a route, in order to have the
     * siteaccess part back in the URI.
     *
     * @param string $linkUri
     * @return string The modified link URI
     */
    public function analyseLink( $linkUri );
}
```

Once modified, the URI is stored in the **semanticPathinfo** request attribute, and the original pathinfo is not modified.