

How to implement a Custom Tag for XMLText FieldType

Version compatibility

This recipe is compatible with **eZ Publish 5.2 / 2013.07**

Custom tags

XMLText fieldtype supports a limited number of tags in its internal eZXML format to render HTML5. However, it is possible to extend the rendering by implementing **custom tags**.

As HTML5 rendering in eZ Publish is done through **XSLT**, you will need to create an XSL stylesheet to extend the rendering.

Note on legacy custom tags

To be able to edit a custom tag from admin interface, you'll still need to [register your custom tag in the legacy kernel](#) (at least the configuration part, template not being mandatory for edition).

- Custom tags
 - Register your custom XSL stylesheet
 - Example of a custom XSL
 - Using Pre-converters
 - Registering a pre-converter
 - Overriding existing XSL templates

Register your custom XSL stylesheet

To activate your custom tag rendering, you need to create an XSL stylesheet and to register it properly:

```
ezpublish.yml
ezpublish:
  system:
    my_siteaccess:
      fieldtypes:
        ezxml:
          custom_tags:
            # Adding ezpublish/Resources/my_custom.xsl (priority defaults
to 0).
            - { path: %kernel.root_dir%/Resources/my_custom.xsl }
            # Adding src/Acme/TestBundle/Resources/another_custom.xsl with
priority 10.
            - { path:
%kernel.root_dir%/../src/Acme/TestBundle/Resources/another_custom.xsl, priority: 10 }
```

Each entry under `custom_tags` is a hash having the following properties:

path

Absolute path to the XSL to import.

Tip

Use `%kernel.root_dir%` parameter to start from `ezpublish/` folder.

priority Priority of your stylesheet in the sequence of importing. The higher it is, the higher precedence it will have.

In XSL imports, in case of template overrides, the last imported XSL always wins. Hence custom XSL are loaded in reverse priority order.

Example of a custom XSL

The following example shows how to render the [YouTube embed custom tag](#) from [jvEmbed legacy extension](#) (see also [related legacy configuration for content](#)).

Note that all selected attributes are in `custom` namespace (this is the case for all custom tags attributes in internal eZXML).

youtube.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://ez.no/namespaces/ezpublish3/xhtml/"
  xmlns:custom="http://ez.no/namespaces/ezpublish3/custom/"
  xmlns:image="http://ez.no/namespaces/ezpublish3/image/"
  exclude-result-prefixes="xhtml custom image">

  <xsl:output method="html" indent="yes" encoding="UTF-8"/>

  <!-- Template below will match the following custom tag: -->
  <!-- <custom name="youtube" custom:video="//www.youtube.com/embed/MfOnq-zXXBw"
  custom:videoWidth="640" custom:videoHeight="380"/> -->
  <xsl:template match="custom[@name='youtube']">
    <div class="jvembed jvembed-youtube">
      <iframe>
        <xsl:attribute name="width"><xsl:value-of
select="@custom:videoWidth"/></xsl:attribute>
        <xsl:attribute name="height"><xsl:value-of
select="@custom:videoHeight"/></xsl:attribute>
        <xsl:attribute name="src"><xsl:value-of
select="@custom:video"/></xsl:attribute>
        <xsl:attribute name="frameborder">0</xsl:attribute>
        <xsl:attribute name="allowfullscreen"/>
      </iframe>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

Tip

PHP functions are registered in the XSLTProcessor, so you can use PHP functions and static method calls to enhance the XSLT process (using `php-function` and `php-functionString` XSLT functions).

However, consider using those for [simple use cases](#). For more complex cases, you'll need to register pre-converters instead.

Using Pre-converters

Pre-converters are services that pre-process the internal XML before the XSLT rendering occurs. It can be useful if you need to manipulate the data stored in eZXML.

An example of use is [what is done for internal links](#). Indeed, internal location or content are linked by their `locationId/contentId` which need to be transformed in a usable URI.

Pre-converters receive the whole `DOMDocument` object for the current field. So you can easily do XPath queries and do some DOM manipulation against it.

Registering a pre-converter

All pre-converters need to:

- Implement `eZ\Publish\Core\FieldType\XmlText\Converter` interface.
- Be registered as a service, with `ezpublish.ezxml.converter` tag.

services.yml in a bundle

```
parameters:
  my.ezxml.pre_converter.class: Acme\TestBundle\XmlText\MyPreConverter

services:
  my.ezxml.pre_converter:
    class: %my.ezxml.pre_converter.class%
    arguments: [@someDependency, @someOtherDependency]
    tags:
      - { name: ezpublish.ezxml.converter }
```

MyPreConverter

```
<?php

namespace Acme\TextBundle\XmlText;

use eZ\Publish\Core\FieldType\XmlText\Converter;
use DOMDocument;

class MyPreConverter implements Converter
{
    public function __construct( $someDependency, $someOtherDependency )
    {
        // ...
    }

    /**
     * Does a partial conversion work on $xmlDoc.
     *
     * @param \DOMDocument $xmlDoc
     *
     * @return null
     */
    public function convert( DOMDocument $xmlDoc )
    {
        // Do something on $xmlDoc
        // You can for example walk through the DOM, do XPath queries, add/modify
        attributes...
    }
}
```

Overriding existing XSL templates

As XSL stylesheets apply for the whole resulted DOM, you can of course override existing templates. This is where the `priority` property in configuration takes its sense.

Built-in XSL templates have **0** as priority

Consider the following example to switch from usage of `` to ``:

strong.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://ez.no/namespaces/ezpublish3/xhtml/"
  xmlns:custom="http://ez.no/namespaces/ezpublish3/custom/"
  xmlns:image="http://ez.no/namespaces/ezpublish3/image/"
  exclude-result-prefixes="xhtml custom image">

  <!-- Original template transforms into <b> -->
  <xsl:template match="strong">
    <strong>
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </strong>
  </xsl:template>

</xsl:stylesheet>
```

ezpublish.yml

```
ezpublish:
  system:
    my_siteaccess:
      fieldtypes:
        ezxml:
          custom_tags:
            # Adding a higher priority to ensure built-in template will be
            overridden.
            - { path: %kernel.root_dir%/Resources/strong.xsl, priority: 10
          }
```