Xentriq Documentation

Release -

Thys ten Veldhuis

Jun 18, 2018

Projects:

This site contains documentation about Xentriq (https://github.com/Nentix/xentriq.core).

Xentriq is a Template Rendering System. It allows for developers to create structured projects and provides in better separation of views and functional implementation using PHP.

Not a CMS?

In the first place Xentriq is a Template Rendering System. It allows for developers to create structured MVC projects and provides in better separation for the Model, View and Controller using PHP.

CMS Addon!

To provide for those who do want a CMS and wish to be able to easily manage content, plugins and templates using a CMS, we have created a separate management-addon for Xentriq also available on Github.

CHAPTER 1

Project development

1.1 Getting started

The following sections should help you get up and running with Xentriq quickly. Use these guides to quickly set up a new project for website development.

1.1.1 Prerequisites

To be able to run a Xentriq project the following is required on your deployment machine:

- Linux/Windows
- Apache HTTP Server/NGINX/IIS
- PHP 7.0 or higher
- PHP modules
  - php-intl
  - php-mbstring
  - php-fileinfo

1.1.2 Composer

1. Install Composer
2. Create your project directory
3. Using your command line run:

```
composer create-project nentix/xentriq.project
```

3. To get started with your first templates and building your website see *Project Structure*

4. See *Deployment* for more information to getting your website up and running

1.1.3 Eclipse

Project creation

1. Download the xentriq.core project, this contains everything to setup a basic templating project.
2. Import the project in to Eclipse using File > Import > Existing Projects in to Workspace.
3. Create a new Eclipse project using your preferred mechanism (for example File > New > PHP Project).
4. In the newly created project create a directory named src and in that directory create the directories named public, private and config.
5. In the private folder create the following folder structure; project/content/templates/. In the templates folder you can place all the Xentriq templates.
6. Create a template named index.xpage and fill it with template content. For example;

```
<html>
  <head>
    <title>Xentriq Test Project</title>
  </head>
  <body>
    This is my first Xentriq project. [link] is the link to this page.
  </body>
</html>
```

7. Right-click the project go to Properties > PHP > Include Path and add xentriq.core in the Projects tab.
8. In the Properties window expand the Project Deployment property and select General.
9. Check Enable project deployment. (This does not enable automatic deployment). Select the preferred Mode, input the Path (e.g. /var/www/xentriq-test-project/) to deploy the project to and input the proper settings in the Remote (SSH) section if you've chosen to use the SSH deployment mode.
10. You are now ready to deploy the project. You can do this by right-clicking the project and under Project Deployment select either Deploy or Enable Automatic Deployment.
11. When you select Deploy the project should get deployed to the location you have specified.
12. If you enable automatic deployment you will have to perform a rebuild. After the initial rebuild any incremental changes will be automatically deployed.
13. See *Deployment* for more information to getting your website up and running
14. To get more information on how to continue building your website see *Project Structure*

Eclipse plugins

We have created several plugins for Eclipse allowing easy development of Xentriq projects.

Xentriq

We have created a plugin for Eclipse allowing easy editing of Xentriq templates. The plugin provides in;

- Easy template file creation.
- Autocompletion.
- Hyperlinking (from template to code).

You can install this plugin by opening your Eclipse IDE, selecting Help -> Install New Software and adding https://nentix.io/xentriq/eclipse/updates .

Tiabo

We have also created a 'Web Deployment'-plugin for Eclipse for easy project development and deployment, called Tiabo. This plugin allows you to make separate projects (for example for your addons and project-specific content). The plugin merges your projects and can deploy either via SSH, SFTP or a local copy.

You can install Tiabo by opening your Eclipse IDE, selecting Help -> Install New Software and adding https://nentix.io/tiabo/eclipse/updates .

1.1.4 Standalone

Because not everyone wants to use Eclipse or wants to have a multi-project setup it is possible to install and develoo Xentriq projects without the use of Tiabo.

1. Download the xentriq.core project, this contains everything to setup a basic templating project.
2. Rename the xentriq.core/src directory to anything of your liking, for the purpose of this example we will use xentriq-test-project
3. To get started with your first templates and building your website see *Project Structure*
4. See *Deployment* for more information to getting your website up and running

1.2 Project Structure

A basic Xentriq project always follows a basic folder structure. The following section describe the purpose of each of these folders in more detail and provides information about where which files of your project should be placed.

Folder structure

- *addons*
- *config*
- *content*
  - *templates*
  - *locale*
- *data*
- *public*
- *vendor*

1.2.1 addons

This folder contains Xentriq addons. An addon provides additional functionality and acts as a backend for your templates.

For more information see *Addon development*.

1.2.2 config

This folder contains configuration files such as the project.json file.

For more information about configuring Xentriq projects see *Configuration*.

1.2.3 content

This folder will contain most of the files of the website such as the templates, javascript, CSS/Sass and more.

templates

This subfolder of contents should contain all Xentriq templates.

For more information regarding templates see *Templating*.

locale

This optional folder can hold localization/translation files used by the localization feature of Xentriq.

For more information see *Language & localization*.

1.2.4 data

This folder is used by Xentriq to store cache files and compiled templates.

Note: This folder should be writable by the user that is used by the PHP process.

1.2.5 public

This is the public folder of your Xentriq project website and the folder which should be the website root in your webserver set-up. This folder also contains the `index.php` file to which all requests should be redirected.

For more information see *Deployment*.

You can include any file in this folder that should be freely accessible. Some examples:

- favicon
- images
- javascript
- css

1.2.6 vendor

This is the Composer vendor folder. It contains the dependent library that your project uses.

1.3 Configuration

1.3.1 Project Configuration

Place configuration in `config` folder with a `.json` file extension. For example; `my-project.json`

```
{
  "name": "MyProject",
  "version": "1.0",

  "config": {
    "debug": false,
    "disableOutputBuffer": false,
    "disableGzip": false
  },

  "bundles": {
    "project-css": {
      "type": "css",
      "content": [
        "css/default.css",
        "css/project.css",
                    ]
                },
                "project-js": {
      "type": "js",
                        "content": [
                                "/js/default.js",
                                "/js/project.js",
                        ]
                }
        }
}
```

JSON structure

config

Type	Description
`debug`	Increases verbosity at the cost of performance, if set to true.
`disableOutputBuffer`	Xentriq uses output-buffering by default. Use this value to disable it
`disableGzip`	Xentriq uses Gzip-encoding by default. Use this value to disable it

bundles

. . . TODO

1.3.2 Environment based Configuration

1.4 Language & localization

Xentriq provides a way for you to customize the content of your website based on the locale of the user.

1.4.1 Language detection

The locale of the user is determined as follows:

1. Using the Accept-Language header
2. Using the value of a cookie named `language` (overrides the header)

In the cookie the locale can be specified in the following formats (case-insensitive):

- en
- en-us
- en_us

1.4.2 Language files

Language files should be located in the `content/locale` folder (see *Project Structure*). A language file should be a JSON file named after the locale which it concerns (for example en.json or en-us.json).

A typical file would be structured as follows:

```
{
  "title": "Your Xentriq project title",
  "heading.example": "Example",
   ...
}
```

Note: The value for each key can contain ICU strings and are therefore parametric.

1.4.3 Localization of templates

The localized text defined in the language files as key value pairs can be included in the template by the use of language tags.

See /templates/language for more details.

1.5 Deployment

1.5.1 Apache HTTP Server

This assumes that Apache HTTP server and PHP have been installed on the machine and that both of them are configured to work with each other.

1. Ensure that your project is deployed to a folder that can be accessed by the web server.
2. Create a virtualhost configuration file such as shown below for your project, point the document root to the 'public' folder.

```
<VirtualHost *:80>
  ServerName xentriq-test-project.local
  DocumentRoot /var/www/xentriq-test-project/public/

  <Directory "/var/www/xentriq-test-project/public/">
    Require all granted

    Options +FollowSymLinks
    AllowOverride All
    AddType application/x-httpd-php .php

    DirectoryIndex index.php
  </Directory>
</VirtualHost>
```

3. Reload your webserver configuration and you should be able to access the rendered template via xentriq-test-project.local

1.5.2 Nginx Server

This assumes that NGINX and PHP-FPM have been installed on the machine and that both of them are configured to work with each other.

1. Ensure that your project is deployed to a folder that can be accessed by the php-fpm linux user.
2. Create a configuration file such as shown below for you project, point the document root to the 'public' folder. Note the configuration section regarding PHP-FPM as this is dependent on your Linux distribution and needs to be customized for your installation.

```
server {
      listen 80 default_server;
      listen [::]:80 default_server;

      root /var/www/xentriq-test-project/public/;

      # Add index.php to the list if you are using PHP
      index index.php index.html index.htm;

      server_name xentriq-test-project.local;

      location / {
              # First attempt to serve request as file, then
              # as directory, then route to index.php for Xentriq.
              try_files $uri $uri/ /index.php$is_args$args;
```

(continues on next page)

(continued from previous page)

```
        }

        # Pass the PHP scripts to PHP-FPM
        location ~ \.php$ {
                include snippets/fastcgi-php.conf;

                # Check you PHP-FPM configuration to find out on which port or␣
↪socket it is listening:
                # PHP-FPM listening to port:
                # fastcgi_pass 127.0.0.1:9000;
                # PHP-FPM configured to listen to socket:
                # fastcgi_pass unix:/run/php/php7.0-fpm.sock;
        }

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        location ~ /\.ht {
                deny all;
        }
    }
```

3. Reload your webserver configuration and you should be able to access the rendered template via xentriq-test-project.local

1.5.3 Internet Information Services (IIS)

Dependencies

To setup Xentriq in IIS the following should be set-up for IIS using the Web Platform installer :

1. PHP Manager for IIS
2. PHP 7.X (NOT FOR IIS EXPRESS)
3. URL Rewrite

Note: Under newer versions of Windows the installer for PHP Manager will not install as it does not recognize IIS 10. To fix this the following can be done:

1. Install .Net Framework 3.5 (this is required by the PHP Manager)
2. Open `regedit`
3. Navigate to `HKLM/System/CCS/Services/W3SVC/Parameters/MajorVersion`
4. Verify that the value is set to 0xA (hex value for 10) and set this value to 9 (or 8)
5. Restart the machine (alternatively it might also work to restart IIS and close all related windows)
6. Use Web Platform installer to install the PHP Manager and PHP 7.x.
7. Chang the regedit value back to 0xA / 10

More information on this issue can be found here

Website setup

1. Deploy the project into a folder on the filesystem (i.e. `C:\\inetpub\\xentriq-test-project.local`)
2. Create a new Website in IIS and point to the `public` directory of the Xentriq project
3. Add the following `Web.config` file to the `public` directory of the Xentriq project. To enable redirection of all URLs to the Xentriq `index.php`

```
<?xml version="1.0" encoding="UTF-8"?>
  <configuration>
    <system.webServer>
        <rewrite>
            <!--Xentriq requires all requests to go to the index.php file.␣
↪This is needed so that additional processing of the url segments can be␣
↪done. Please keep below rewrite rule intact. -->
            <rules>
                <rule name="Redirect all to index.php">
                    <match url=".*" />
                    <action type="Rewrite" url="index.php" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

Troubleshooting

Form tags are not rendering/rendering as 'Unable to generate security token'

This is caused by the openssl.cnf file not being found in the PATH of the server. Ensure that the correct file is part of the PATH or that the OPENSSL_CONF environment value is set to the path of the *openssl.cnf* file.

CHAPTER 2

Templating

This section contains an overview of the templating syntax of Xentriq. In the use-cases/index section some more practical examples are shown that will help you get started in building a maintainable and modular website.

2.1 File Types

There are two file types that Xentriq uses. Each type has its specific use case:

Type	Description
`.xpage`	Is accessible as a page/through a request and can be included in any template.
`.xpart`	Can only be included in other templates using *include* or *implement* tags and is not accessible as a page.

2.2 Tags

2.2.1 Actions

An action can be used to generate a link that triggers a function in an Addon (see *Addon development*).

This is very useful for use in AJAX calls through JavaScript.

See ../use-cases/actions-and-ajax for more information about this use case.

Example

```
[action:<addon>-><function> arg1='<argument>']
```

This will output a link, which when called will trigger a call to the specified function of the addon.

It is also possible to get the urls as *Variables*:

```
[var1=action:<addon>-><function> arg1='<argument>']

[var1]
```

2.2.2 Count

The count block is similar to a traditional for-loop.

This can be useful if you need to use numbering for rendering certain template structures or when calling Addon functions (see *Addon development* for more details).

```
[count 1 to 10 as number]
   [number]
[/count]
```

The condition used for the loop is `number < limit`.

Therefore, this example will output the numbers: **1, 2, 3, 4, 5, 6, 7, 8, 9**.

2.2.3 Foreach

The foreach block can be used to loop through the contents of an array-variable.

This is often used to show a list of a certain set of items. See ../use-cases/lists-and-foreach

```
[foreach <vars> as <var>]
   [var]
[/foreach]
```

Additional properties

provide/describe

It is often useful to also have information about the current index of the current item. This is possible by using the additional keyword `provide` or `describe` as shown in the following example:

```
[foreach <vars> as <var> provide <metaVar>]
   ...
[/foreach]
```

Inside the block you can then use the *metaVar* to retrieve additional information about the current state of the loop:

Property	Description
`size`	Is accessible as a page/through a request
`index`	The index in the array of the current item.
`isFirstItem`	Indicates whether the current item is the first in the array.
`isLastItem`	Indicates whether the current item is the last in the array.

2.2.4 Form

A very common use case when developing a website is the submission of Forms. In Xentriq it is possible to easily POST/GET the contents of a form to a function provided in an Addon (see *Addon development*).

The following is an example of how the Form-tag can be used to achieve this:

```
[form:<addon>-><function> arg1='<argument>' method="POST"]

[/form:<addon>-><function>]
```

The above tag will be rendered in your html to a normal `<form>` HTML-tag with the action pointing to a URL that will trigger the appropriate function.

```
<form action="<Xentriq generated URL>" method="POST">

</form>
```

Providing HTML attributes

Often it is desirable to provide additional HTML attributes to the `<form>`-tag that is rendered. The list below shows the attributes Xentriq will render to html if they are provided to the [form] tag as argument. Any other argument will only be passed to the Addon method.

Arguments rendered to HTML

- accept
- accept-charset
- action
- autocomplete
- enctype
- method
- name
- novalidate
- target
- onsubmit
- accesskey
- class
- contenteditable
- contextmenu
- dir
- draggable
- dropzone
- hidden

- id
- lang
- spellcheck
- style
- tabindex
- title
- translate

2.2.5 Variable arguments & modifiers

You can apply an argument or modifier to a variable to format the output;

```
[var <modifier1>="<modifier1-value>" <modifier2>]
```

Modifications

General

Modifier	Description
`lowercase`	Converts the variables string value to lowercase
`uppercase`	Converts the variables string value to uppercase
`ucfirst`	Converts the first letter of the variables string value to uppercase
`negative`	Negates the variable (*true* becomes *false*)
`sizeof`	Get the size of the variable. (array or string length)

Encoding

Modifier	Description
`json`	Get the variable as a json encoded string
`urlencode`	Get the variable as a url encoded string
`urldecode`	Get the variable as a url decoded string
`base64encode`	Get the variable as a base64 encoded string
`base64decode`	Get the variable as a base64 decoded string
`utf8encode`	Get the variable as a utf-8 encoded string
`utf8decode`	Get the variable as a utf-8 decoded string
`htmlspecialchars`	Get the variable with all special HTML characters encoded. This is equivalent to the PHP htmlspecialchars function.

Modifiers with arguments

The following modifiers are to be used with an argument (for example `[var modifier="argument"]`).

Modifier	Description
`default`	If the variable does not have a value (`== null`) the arguments' value will be used.
`divide`	Divides the variable by the provided argument.
`dateformat`	Formats the provided variable as a date with the format provided in the argument The variable can hold a date in the form of a string or of a Unix timestamp in seconds or milliseconds. For more information about formatting see PHP date
`join`	This can be used to join the values in the array variable together. The argument provided will be used to join the values together. For example if `arrayVar` has value `['a','b']` and `[arrayVar join=", "]` is used the resulting output will be the string `a,b`.

2.2.6 Include & Implement

The *include* and *implement* tags described below can be used combine different templates to achieve code-reusage and create re-usable components.

For a more concrete example on using these tags effectively see /templates/use-cases/reusable-components.

Include

Include can be used to embed another template into the current template. This feature is often used in combination with .xpart files.

When including a template it is also possible to pass arguments to the included template.

This can be used as follows:

```
[include:'<template>' arg1='<argument>']
```

Implement

The implement-tag can be used to wrap the current template inside another template. This is especially useful if many templates share common HTML-structures.

A frequent use case is to have a shared layout containing for example a header, footer and menu and have each template implement that layout to prevent having to duplicate the template code.

If the template to be implemented looks like this:

```
<div class="wrapper">
  [part:'default' /]
 </div>
```

And the template that should implement the layout looks like this:

```
[implement:'wrapper']
   [part:'default']
      The content to be placed inside the 'default part'
   [/part]
[/implement]
```

The resulting rendered template will be:

```
<div class="wrapper">
  The content to be placed inside the 'default part'
 </div>
```

[part='<part-name>']

Note in the above example the use of the `part-tag`. In the implemented template this tag indicates the location at which the content of the part-tag in the implementing template should be rendered to.

Note: It is also possible to have multiple part tags with different names to inject content into different parts of the template that is implemented.

2.2.7 Language

For localization purposes Xentriq supports the use of language files in combination with language tags to make it easy to develop a multi-lingual website.

This section describes the use of language tags throughout the templates. To learn more about the language files see *Language & localization* in the *Project development* section.

A language tag can be identified by the use of curly braces `{ }` instead of square brackets. The content of the tag is the key that is used in the localization file.

If the localization file for example contains the following JSON structure:

```
{
  "translation.key" : "Translation value"
}
```

The way to output the text `Translation value` will be as follows:

```
{translation.key}
```

Message Formatting

The localization funcitonality of Xentriq is based on the ICU library. This means that you can also do formatting with variables in your language file in combination with languate tags.

If the localization file for example contains the following JSON structure:

```
{
  "translation.key" : "Translation value with {variable}"
}
```

The way to output the text `Translation value with EVERYONE` will be as follows:

```
{translation.key variable="EVERYONE"}
```

For more information about ICU message formatting and the PHP intl library see:

- ICU Formatting Messages Documentation
- PHP Internationalization Functions Documentation

2.2.8 Link

By using the link-tag it is possible to retrieve the current url (e.g. http://localhost/current/path)

```
[link]
```

The link tag can also be used to generate a link to another template based on a path rooted at the templates directory.

```
[link:'/path/to/page']
```

In this case the 'page'-template is located at /content/templates/path/to/page.xpage

For more details about project layout see *Project Structure*

2.2.9 Variables

```
[form:<addon>-><function> arg1='<argument>']

[/form:<addon>-><function>]
```

```
<form>

</form>
```

CHAPTER 3

Addon development

This is an overview of the syntax that can be used for creating templates that can be rendered by Xentriq.

3.1 Tags

Addons can add several types of tags to implement and extend functionality.

3.1.1 Action

The *action* tag is structured as followed.

```
[action:<addon>-><function> arg1='<argument>']
```

So having the following implementation in PHP.

```
class ExampleAddon
{
     function templateActionFunction( TemplateTag $tag )
     {
          // TODO example implementation
     }
}
```

Assuming the name for the addon would be *example*. Would result in the tag;

```
[action:example->templateActionFunction arg1='<argument>']
```

Unlike regular functions, this function will not be called when the template is rendered. The *action* tag will generate a link. Navigating to this link will result in the function being called.

3.1.2 Form

The *form* tag is structured as followed.

```
[form:<addon>-><function> arg1='<argument>']

[/form:<addon>-><function>]
```

So having the following implementation in PHP.

```
class ExampleAddon
{
     function templateFormFunction( TemplateTag $tag )
     {
          $arg1 = $tag->fetchArgument( 'arg1' );
          $input = Request::input( 'name' );

          $success = !empty( $input );
          if( !$success )
          {
               Utils::registerError( 'You have to tell us your name..' );
          }

          /**
            * if $success is false, the user will be redirected back to the page␣
↪containing the form or the path given in the `errorDestination` argument.
            * if $success it true, the user will be redirected to the the page␣
↪containing the form or the path given in the `successDestination` argument.
            *
            */
          Request::redirectAfterAction( $success );
     }
}
```

Assuming the name for the addon would be *example*. Would result in the tag;

```
[form:example->templateFormFunction arg1='<argument>' onsubmit=
↪"javascript:formHandler();" successDestination="/thankyou"]
      [foreach errors as error]
            [error]
      [/foreach]

      <input type="text" name="name" value="" />

      <input type="submit" value="Submit" />
[/form:example->templateFormFunction]
```

The *form* tag will generate a html form. Arguments matching valid DOM arguments will be rendered in the html tag (Eg. method, onsubmit), other arguments will be omitted and passed to the function when it is called.

Unlike regular functions, the function for a form tag will not be called when the template is rendered. Submitting the form will result in the function being called.

The generated html will look something like this

```
<form onsubmit="javascript:formHandler();">
      <input type="hidden" name="do" value=
↪"E44A35E7687A3766116423A2960FE26D309BF92074F4275675B6316357EF9540AE12C6069C6F5D7D14F6F6D1308EF2A70
↪" />
```

(continues on next page)

(continued from previous page)

```
    <!-- Depending if the user omitted the value 'You have to tell us your name..'␣
↪could be displayed here. -->

    <input type="text" name="name" value="" />

    <input type="submit" value="Submit" />
</form>
```

3.1.3 Function

Having the following implementation in PHP.

```
class ExampleAddon
{
    function templateFunction( TemplateTag $tag )
    {
        // TODO example implementation
    }
}
```

Assuming the name for the addon would be *example*. Would result in the tag;

```
[example->templateFunction arg1='<argument>']

[/example->templateFunction]
```

This function will be then called when the template is rendered.

3.2 Hooks

Addons can hook in to existing functionalities provided by Xentriq or any other addon and even provide their own hooks to allow other addons to benefit from their functions.

Read more about hooks on the following pages.

3.2.1 Hooks

Addons can hook in to existing functionalities provided by Xentriq or any other addon and even provide their own hooks to allow other addons to benefit from their functions.

Default

Xentriq has several hooks integrated and called by default.

Hook	Description
`xentriq.bootstrap.route`	Called before any routing and/or rendering is executed.

Implementation

A function can be executed when a hook is called by configuring the hook in the addon's configuration file.

```
{
    "name": "Example Addon",
    "version": "1.0",

    "provides": [{
        "identifier": "example",
        "clazz": "Example\\ExampleAddon",
        "hooks": {
            "xentriq.bootstrap.route": "routeHookFunction"
        }
    }]
}
```

```
namespace Example;

class ExampleAddon
{
    function routeHookFunction()
    {
        // TODO example implementation
    }
}
```

Creating a new hook

Addons can add additional hooks by just calling them in the place you wish to execute any configured functions.

```
Addons::allowHook( 'custom.hook.id' );
```