
Novius OS Documentation

Release Chiba 1.0.1

Novius

November 22, 2013

Contents

Welcome to Novius OS documentation. It is hosted and generated by [Read The Docs](#).

All contributions are welcome: Reporting or fixing errors, submitting improvements or translations.

Sources are on [Git Hub](#), we look forward to your Pull Request. [Contact us](#) if you need help with your contribution.

- [API documentation](#)
- [Version française](#)
- [Japanese version](#)

Table of contents

1.1 Install Novius OS

1.1.1 Installation

Contents

- Installation
 - General requirements
 - Quick installation
 - Installation via Zip file
 - Advanced installation

General requirements

- **Have a LAMP server up and running with PHP 5.3+.**

```
sudo apt-get install apache2 php5 mysql-server libapache2-mod-php5 php5-mysql
```

- **Apache mod_rewrite is enabled.**

```
sudo a2enmod rewrite
```

These commands are given as an example if you install Novius OS on your local machine or a server for which you have admin rights.

They are Ubuntu commands, adapt them to your distribution if necessary.

Note: In theory Novius OS can be installed on other servers than **Apache**.

Quick installation

Requirements

- Have a command line access to the server and being granted **sudo** admin rights.
- **Git** is installed.

Installation

Open a terminal and enter:

```
cd /var/www
sudo wget http://raw.githubusercontent.com/novius-os/ci/master/chiba/tools/install.sh && sh install.sh
```

Once the installation completes:

- Open your browser at <http://your.domain/novius-os/> (replace `novius-os` with the directory name you've chosen).
- Follow the steps of the *setup wizard*.

Note:

- For a local installation, the URL is probably <http://localhost/novius-os/>.
 - If your server's `DOCUMENT_ROOT` isn't `/var/www/`, change the first command accordingly.
-

Installation via Zip file

We recommend you follow this procedure when installing Novius OS on shared hosting:

- Download [novius-os.chiba.1.0.1.zip](#).
- Unzip the file.
- Upload (or move) the `novius-os` directory to your server's `DOCUMENT_ROOT` (using FTP for instance).
- Open your browser at <http://your.domain/novius-os/> (replace `novius-os` with the directory name where Novius OS has been unzipped).
- Follow the steps of the *setup wizard*.

Advanced installation

Configure a Virtual Host

The following commands are provided as example when installing Novius OS on Ubuntu, you should adapt depending on your distribution.

```
sudo nano /etc/apache2/sites-available/novius-os
```

Replace **nano** with any text editor.

Replace `novius-os` with the name you want for your Virtual Host.

Copy the following configuration in the file you just opened and save.

Change the line `ServerName` with your domain name when installing on a live server.

Likewise, change `/var/www/novius-os` with the folder you installed Novius OS into.

```
<VirtualHost *:80>
    DocumentRoot /var/www/novius-os/public
    ServerName    novius-os
    <Directory /var/www/novius-os/public>
        AllowOverride All
        Options FollowSymLinks
    </Directory>
</VirtualHost>
```

The default configuration has a `public` folder. This is where the `DocumentRoot` should point.

Enable the new `VirtualHost`:

```
sudo a2ensite novius-os
```

Then, reload **Apache** to apply the new configuration.

```
sudo service apache2 reload
```

Configure the `hosts` file, when installing on your computer If you install Novius OS on your local computer, you must add a line in the `/etc/hosts` file, containing the value you entered for `ServerName` (`novius-os` in the above example).

```
sudo nano /etc/hosts
```

Add the following line:

```
127.0.0.1    novius-os
```

Advance installation with Git

You should clone the repository available on GitHub:

```
git clone --recursive git://github.com/novius-os/novius-os.git
```

This command downloads the main repository, and its submodules :

- `novius-os` : Novius OS core, which contains submodules itself (like `fuel-core` or `fuel-orm`).
- Several submodules in `local/applications`: `blog`, `news`, `comments`, `form`, `slideshow` and other applications.

The repository default branch is latest stable version of Novius OS.

New versions will be made available in new branches.

For now, every dependent repository of `novius-os/novius-os` share the exact same version number. It means that any application available on our Github exists in the same versions as the core. So if you're using the `chiba` version of `novius-os/core`, then you should also use `novius-os/app` in the same `chiba` version number.

After the initial `clone`, if you want to change the Novius OS version you're using, don't forget to update the submodules!

Here's how to use the latest *nightly* (it's in the `dev` branch):

```
cd /var/www/novius-os/  
git checkout dev  
git submodule update --recursive
```

1.1.2 Setup wizard

You've followed the *first part of the installation process*. Novius OS files are now installed, the server is set-up and you access <http://your.domain/novius-os/>. You've done the hardest, now comes the easy part :-).

Step 1: Checking requirements

This step should be straightforward if you've installed Novius OS following the quick procedure. In other cases, don't worry if a lot of red comes up! Your website just needs writing permissions in some directories. This step gives you the explanations and the commands to run to fix everything.

Step 1 / 4 : checking pre-requisite
Please note [a summary](#) of the commands is available below

PHP configuration directive <code>short_open_tag</code> = On	OK
PHP configuration directive <code>magic_quotes_gpc</code> = Off	OK
APPPATH/config/ is writeable by the webserver <small>This is required temporarily to write the <code>db.php</code> and <code>crypt.php</code> config files</small> <code>chmod a+w /data/www-local/novius-os/local/config</code>	Error
APPPATH/data/ is writeable by the webserver <code>chmod a+w /data/www-local/novius-os/local/data/</code>	Error
APPPATH/data/config/ is writeable by the webserver	Error

To fast-track this step, just copy the commands' summary given at the bottom of the page and copy them in a terminal. You're done!

Step 1 / 4 : checking pre-requisite

PHP configuration directive short_open_tag = On	OK
PHP configuration directive magic_quotes_gpc = Off	OK
APPPATH/config/ is writeable by the webserver	OK
APPPATH/data/ is writeable by the webserver	OK
APPPATH/data/config/ is writeable by the webserver	OK
APPPATH/data/media/ is writeable by the webserver	OK
APPPATH/cache/ is writeable by the webserver	OK
APPPATH/cache/media is writeable by the webserver	OK

Step 2: Setting up the MySQL database

You need a MySQL database including a user with the required rights. For shared hosting, your provider must have given you these details. In other cases, here is an example for a localhost database:

```
CREATE DATABASE `your_db_name` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON `your_db_name`.* TO 'your_user_name'@localhost IDENTIFIED BY 'your_password';
FLUSH PRIVILEGES;
```

Fill in these four fields according to your database configuration.




Step 2 / 4

Configuring the MySQL database

This will create two files `local/config/db.php` and `local/config/crypt.php` and—more importantly—the tables needed for Novius OS to run.

Step 3: Creating the first administrator account



Step 3 / 4

Create the first administrator account

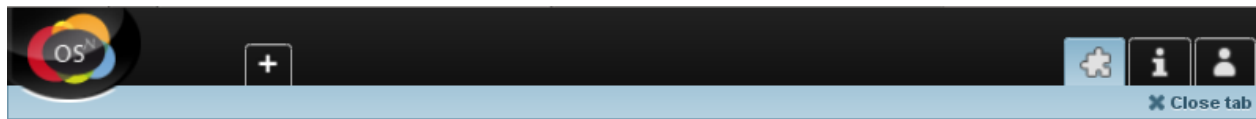
Step 4: Finishing the installation



Warning: We urge you to follow the recommendation of this page.
On setting contexts, refer to the *principles* and *API documentation*.

Applications

You're redirected to the applications manager. Install the applications you need.



Novius OS framework configuration

Up to date!

Local configuration

Up to date!

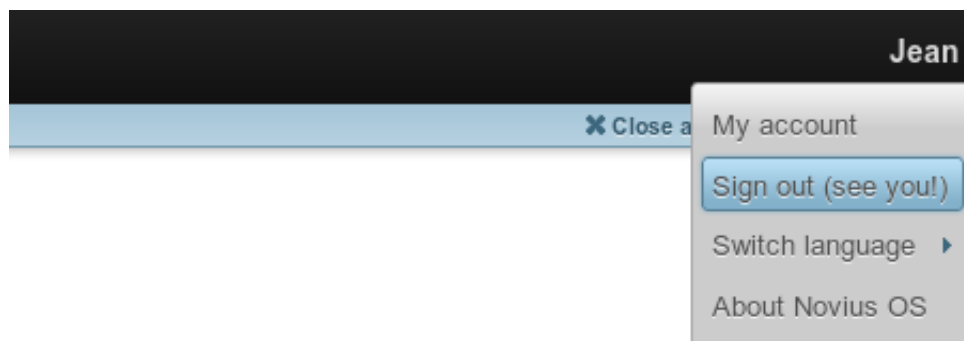
Applications

Installed and ready to use	Actions
✳ Novius OS Basic Templates	↓ Uninstall

Available for installation	Actions
Blog	↑ Install
Blog / News (required for Blog or News)	↑ Install
Comments (required for Blog or News)	↑ Install
News stories	↑ Install
Simple Facebook share	↑ Install
Simple Twitter share	↑ Install

Signing in and out

To sign out, click your name in the top-right corner. A menu shows:



You're redirected to the sign-in form.

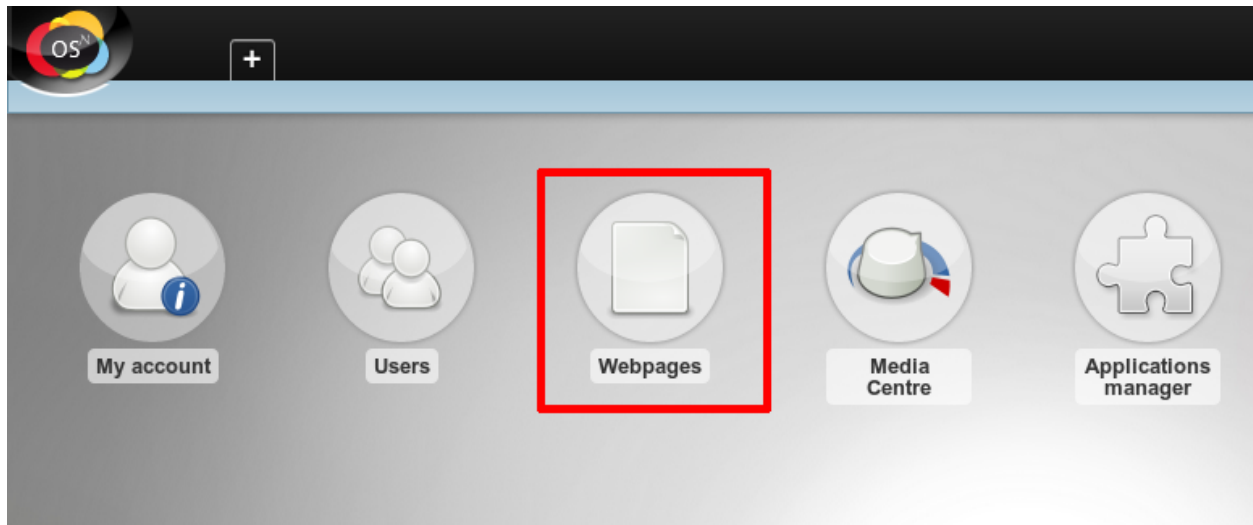


1.1.3 What's next

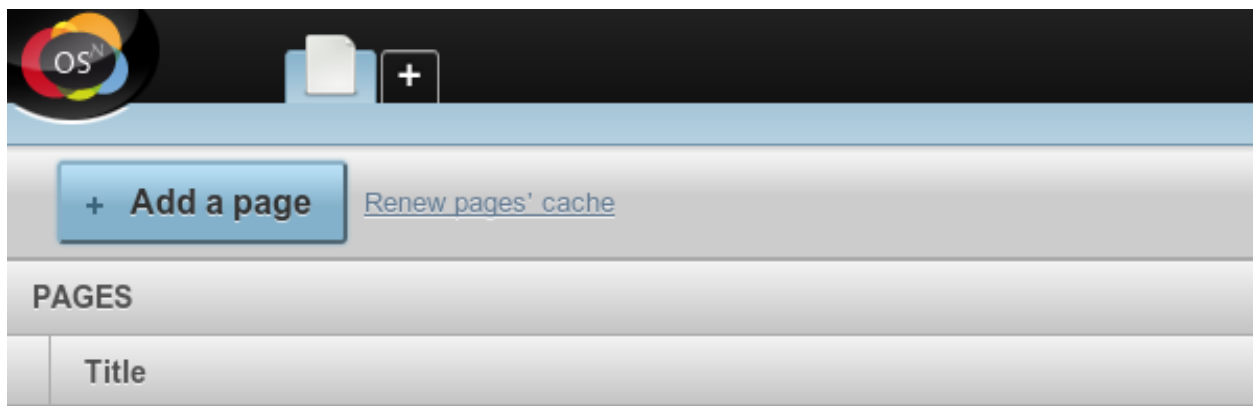
First page

Open the Webpages application

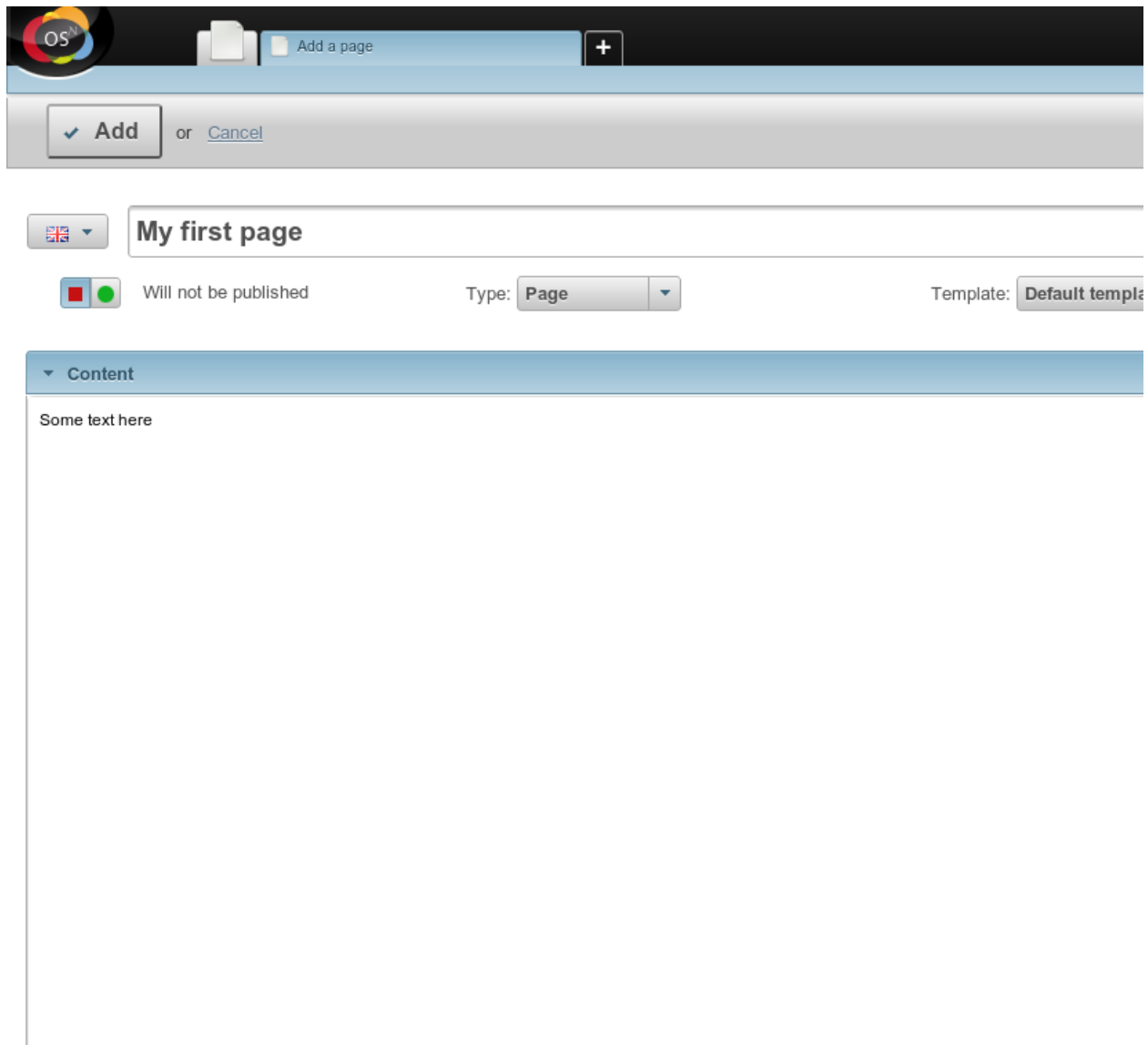
If you want create a website, go to the **Webpages** application:



Add your first page



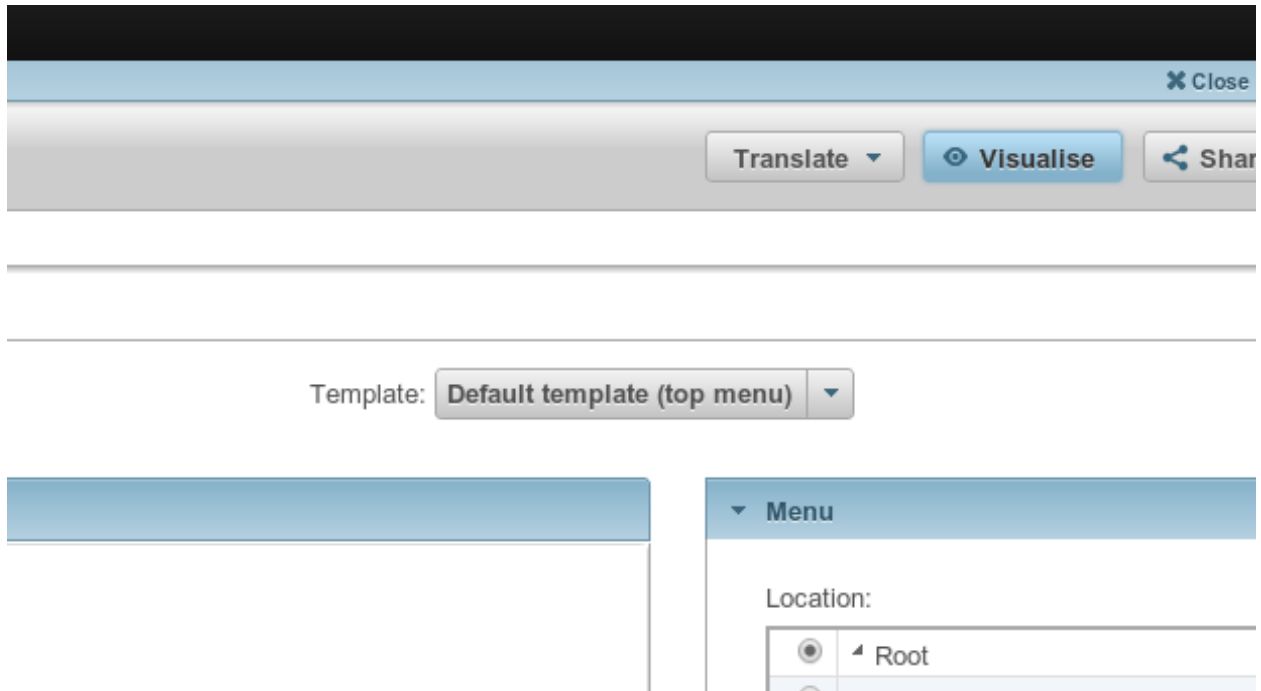
Write some content and hit ‘Add’



The screenshot shows the Novius OS page editor interface. At the top, there is a dark header bar with the Novius OS logo on the left and a 'Add a page' button with a plus icon on the right. Below the header is a light blue bar containing a '✓ Add' button and a 'Cancel' link. The main editing area has a title field containing 'My first page' and a language dropdown menu showing a flag icon. Below the title, there are three status indicators: a red and green square icon, the text 'Will not be published', a 'Type: Page' dropdown menu, and a 'Template: Default template' dropdown menu. The main content area is titled 'Content' and contains the text 'Some text here'.

Preview your work

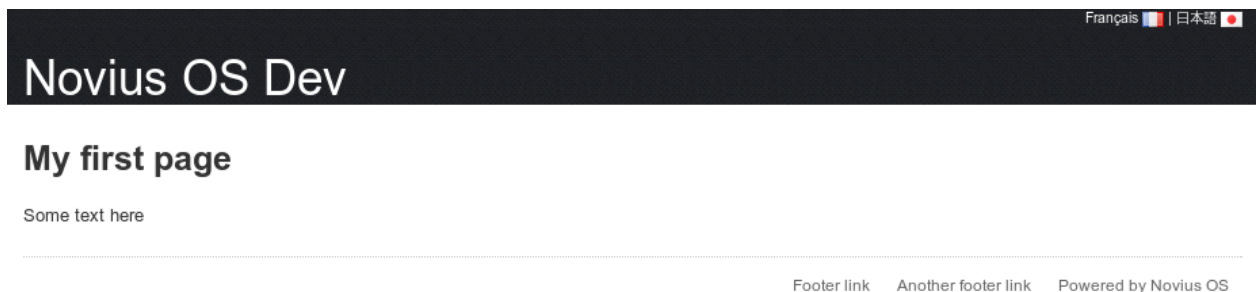
The **Visualise** action allows you to preview the page before publishing it.



Publish your page

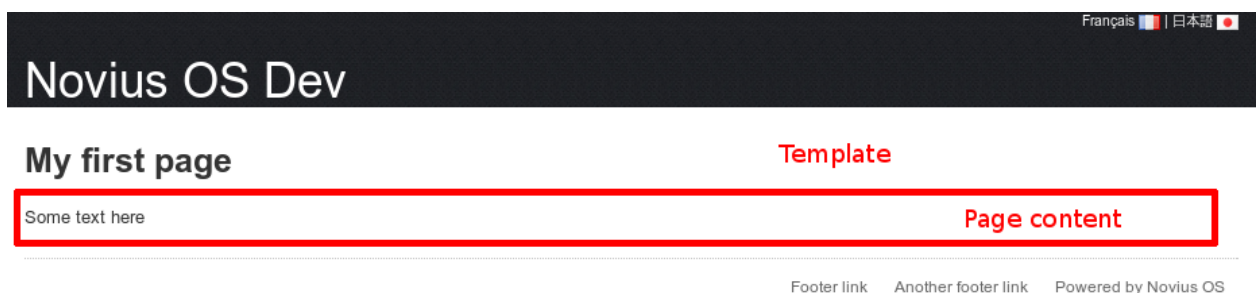
Once you're happy with your work, choose 'Will be published' and save.

Look in awe at what you've achieved:

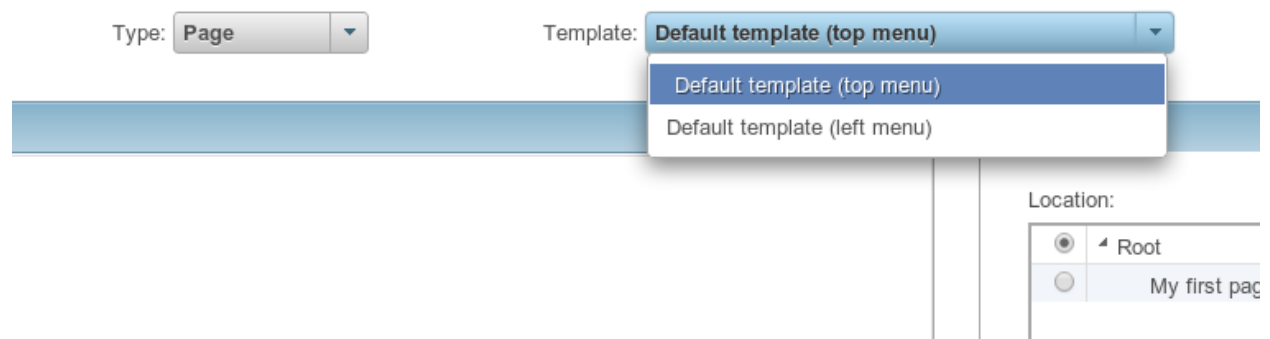


Templates

Pages need templates. They contains the content written in the back-office and set the style.



The template is chosen when adding or modifying a page.



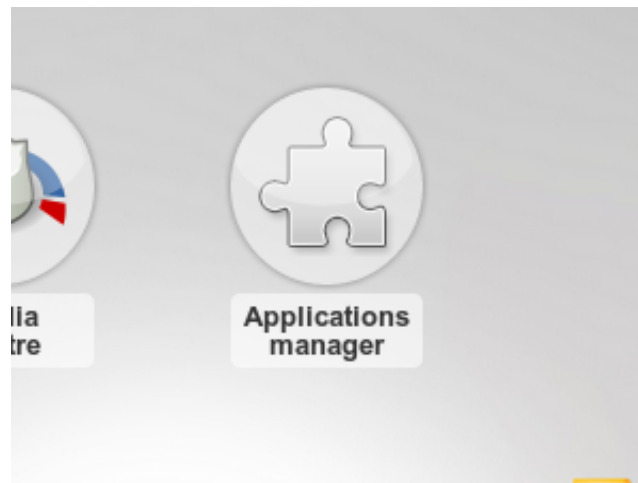
New templates can be added. Templates are embedded in the applications. To add a template, one must therefore add an application (with the *applications manager*).

Applications

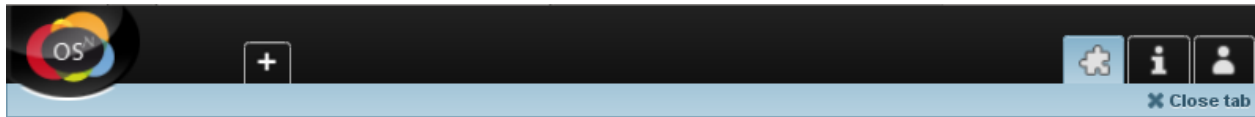
Thanks to the applications, you can add new features to Novius OS.

The applications manager

Allows you to install and uninstall applications.



After modifying the `metadata` of an application or your website (Novius OS' instance), you must apply the changes in the applications manager. This is also the case when a native application has been modified.



Novius OS framework configuration

Up to date!

Local configuration

Up to date!

Applications

Installed and ready to use	Actions
✳ Novius OS Basic Templates	↓ Uninstall

Available for installation	Actions
Blog	↑ Install
Blog / News (required for Blog or News)	↑ Install
Comments (required for Blog or News)	↑ Install
News stories	↑ Install
Simple Facebook share	↑ Install
Simple Twitter share	↑ Install

1.1.4 Updates

Updating the files

Git

If you installed Novius OS with **Git**, run these commands from Novius OS directory:

```
git fetch origin
git checkout master/chiba
git merge origin/master/chiba
git submodule update --recursive --init
```

Zip

If you downloaded the Zip file, the procedure is more complex.

- Let's say you installed novius OS in `my_site/`.
- First, backup your old directory (copy it and make a .zip file with it).
- Download the [new Zip of Novius OS](#) and extract it. You now have a `novius-os/` directory.
- In `my_site/`, delete the following directories:

- my_site/novius-os/
- my_site/local/migrations/
- Every my_site/local/applications/noviusos_* directories

- **Copy the following directories and files from novius-os/ to my_site/ :**

- novius-os/
- local/migrations/
- Every local/applications/noviusos_* directories
- public/install.php
- public/.htaccess

You may need .sample files later in the procedure. If so, copy them from the novius-os/ to my_site/ when asked.

Now you can continue the update.

Run the migration

Before running the automated migration tools, please backup your database.

If you're allowed to access **SSH** on the server, run this command from your Novius OS directory:

```
sudo php oil refine migrate
```

If you can't access **SSH**, you can run the migration from your browser:

- First, you need to rename the public/migrate.php.sample file to public/migrate.php.
- Open the file in your browser, such as http://www.my_site.com/migrate.php.

Applications Manager

From the back-office of Novius OS, open the Applications manager and update the applications which need it.

Migrate your developments

If you have personal developments, you need to follow the *Migration guide from the 0.2 version to the Chiba 1 version*.

1.1.5 Post-install optimisations

Configure and enable XSendFile

To understand what XSendFile is and its usefulness, go here : *Media centre*.

Apache

Apache provides the feature through the **mod_xsendfile** module. Then, the `.htaccess` must be updated to enable it:

```
# Post-installation optimisation
<IfModule xsendfile_module>
    XSendFile On

    # Replace "novius-os-install-dir" by the real Novius OS installed directory
    XSendFilePath /novius-os-install-dir/local/data

</IfModule>
```

Novius OS know when the module is enabled automatically and enable the `XSendFile` feature accordingly.

nginx

nginx provides the feature natively, but the header used is `X-Accel-Redirect`. In that case, you need to edit the `config.php` file to tell Novius OS the appropriate header :

```
<?php
return array(
    // ...
    'novius-os' => array(
        // ...
        'use_xsendfile' => 'X-Accel-Redirect',
    ),
);
```

1.2 Discover Novius OS

1.2.1 Software's fundamentals

MVC structure

Novius OS follows the [Model-View-Controller](#) pattern, which defines how to work:

- when designing applications;
- when organising a project running Novius OS.

Frameworks usage

Framework usages influences us a lot when designing and implementing applications, and it's really helpful to know the role they play in. However, this documentation relates to Novius OS, so please refers to external documenations and tutorials for more informations about them.

FuelPHP

[FuelPHP](#) is the PHP framework used by Novius OS.

The most used parts are the ORM, the cascading file system and the data validation. The framework also provides a lot of tools which make application development easier, like the *Arr* <<http://docs.fuelphp.com/classes/arr.html>> class for example.

FuelPHP's ORM

ORM stands for [Object-relational mapping](#).

The ORM allows to handle the database using PHP objects (and classes), allowing to express relations between them. An example is worth a thousand words:

```
$new_monkey = Model_Monkey::forge();
$new_monkey->monk_name = 'Julian';
$new_monkey->save();

$monkeys = Model_Monkey::find('all');
foreach ($monkeys as $monkey) {
    //...
}

$monkey = Model_Monkey::find(4);
$monkey->delete();
```

Novius OS relies on [FuelPHP's ORM](#). Please refer to its documentation.

But Novius OS adds another notable feature to the ORM: *Behaviours*. *Behaviours* allows to extends *Model* in a standardised and reusable way.

They're similar to FuelPHP's [Observers](#) but are more powerful:

- As *Observers*, they have configurable options.
- As *Observers*, they can intercept events to take action on the *Model* (such as *before_save*, triggered before updating the database).
- In addition, they also provide methods (static or instance) on the *Model*.
- They also can provide new events.

jQuery UI / Wijmo

Although actions are done server side using PHP, Novius OS is mostly written in JavaScript. This is explained by the utmost importance given to the UI and the UX (see *UI guidelines*).

To offer rich interfaces and interactions, Novius OS uses several JS frameworks:

jQuery

This framework makes HTML manipulation and DOM much simpler. It's not especially UI-oriented.

[Documentation](#)

jQuery UI

Built on top of jQuery to provide interface interactions. Most of the Novius OS UI comes from this framework.

[Documentation](#)

Wijmo

Built on top of jQuery UI. Provides additional rich-interface elements, called widgets.

[Documentation](#) and [Examples](#)

There's a hierarchy between those frameworks, Wijmo has the most impact on Novius OS' ergonomics.

1.2.2 Folder organisation

In Novius OS

Root directories

- `~/novius-os/`: Novius OS core.
- `~/local/`: Your website
- `~/public/`: `DOCUMENT_ROOT` of the website
- `~/logs/`: logs directory

public directory

A file can either be:

- executable or non-executable
- provided by the developer or provided by the application

This leads to 4 possible usages, and each of them is reflected in the `~/public/` folder:

- `~/public/static/`: equivalent to assets. Non-executable files provided by the developer or Novius OS.
- `~/public/data/`: Non-executable files generated by Novius OS.
- `~/public/htdocs/`: Executable files provided by the developer or Novius OS
- `~/public/cache/`: Executable files generated by Novius OS.

Note: Here, Novius OS refers to the core and also to any application of the website.

There's a 5th directory `~/public/media/`, used by the *Media centre*.

Where Novius OS can write the developer cannot, and vice-versa.

`~/public/static/` and `~/public/htdocs/` have the same sub-directories structure:

- `~/novius-os/`: Novius OS files.
- `~/apps/<application_name>/`: Files from developers of *applications*.

These directories are symbolic links, created upon Novius OS installation or when activating *applications*.

These symbolic links respectively point to `htdocs` and `static` directories of Novius OS or the application.

See below for the *directories organisation within an app*.

core directories

- `~/novius-os/framework/`: Novius OS framework
- `~/novius-os/fuel-core/`: FuelPHP framework
- `~/novius-os/packages/`: FuelPHP packages

local directory

- `~/local/applications/`: Novius OS *applications*.
- `~/local/cache/`: Contains resized medias.
- `~/local/classes/`: Your PHP classes.
- `~/local/config/`: Your Novius OS configuration files.
- `~/local/data/`: Files generated by Novius OS.
- `~/local/metadata/`: metadata files for your website, generated by Novius OS.
- `~/local/migrations/`: migration classes.
- `~/local/views/`: Your Views files.

Note: `classes` and `views` directories should not contain a lot of files, since most of your developments should go in *applications*.

Within an application

MVC applies to core, and also to applications



There are 6 main directories:

classes It's where your logic belongs, i.e. classes which defines and manipulates data. At least there are controllers and models of the application. We can also find tools used by the views or the controllers.

config

This directory gather all the informations to represent your models. Controllers perform logical operations on your data, but also needs additional data to pass on to the views in order to display them. These informations are separated from controllers (which don't have logical value) and from the views. The latter receives data as parameters and never fetch them on their own.

The configuration file for the `controller/admin/monkey.ctrl.php` controller will be located at `config/controller/admin/monkey.config.php`. A class and its configuration file share a similar naming convention.

lang This directory contains translation files, with a sub-directory for each language.

migrations This directory contains migration files.

static This directory contains JavaScript and CSS files, and public resources (such as images).

views This directory contains files responsible to display the data.

1.2.3 Differences with FuelPHP

Constants path

See *API documentation for constants*.

Autoloader

Two namespaces are added by Novius OS :

novius-os points to *NOSPATH*.

local points to *APPPATH*.

Bootstrap and entry points

In Novius OS, front-office and back-office are two very separated areas.

So instead of a single `index.php` entry point from FuelPHP, Novius OS has two entry points:

- `~/novius-os/htdocs/admin.php`: Back-office entry point. Handles all URL starting with `/admin/`.
- `~/novius-os/htdocs/front.php`: Front-office entry point. Handles all URL ending with `.html` or the root of your website.

Back-office entry point

Novius OS defines a special route for the back-office, which works as follow:

```
<?php
'routes' => array(
    '^admin/(:segment)/(:any)' => '$1/admin/$2',
),
```

For example, the URL `admin/noviusos_page/page/insert_update/113` will be transformed into `noviusos_page/admin/page/insert_update/113`. Which boils down to executing the `action_insert_update` method from the `Controller_Admin_Page` controller of the `noviusos_page` application.

See Also:

FuelPHP's *documentation about routing*.

1.2.4 UI guidelines

Novius OS interface is built upon great ergonomic guidelines. To develop applications, two of them should be known: tab navigation and App desk.

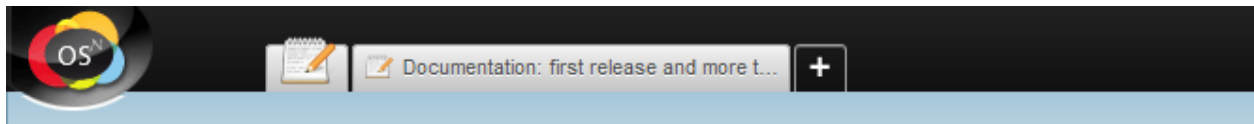
Tabs navigation

[See the screencast related to tabs navigation](#)

Tabs navigation organise the work of the end-user. The purpose is to increase productivity, by limiting repetitive tasks and pages loading.

There are two types of tabs:

- **Application tab** : Application tabs doesn't have a title, but only a big icon of the application. It contains the App Desk (see below).
- **Item tab** : from the application tab, we reach the edition of visualisation of an item, which takes place in a new tab. These kind of tab shows the item's title and a small icon of the application.



There are many advantages to tab navigations. We'll emphasise: - several items of the same application can be edited in parallel; - extremely fast switching between different items; - the user can resume its work where he left out (opened tabs are preserved upon time).

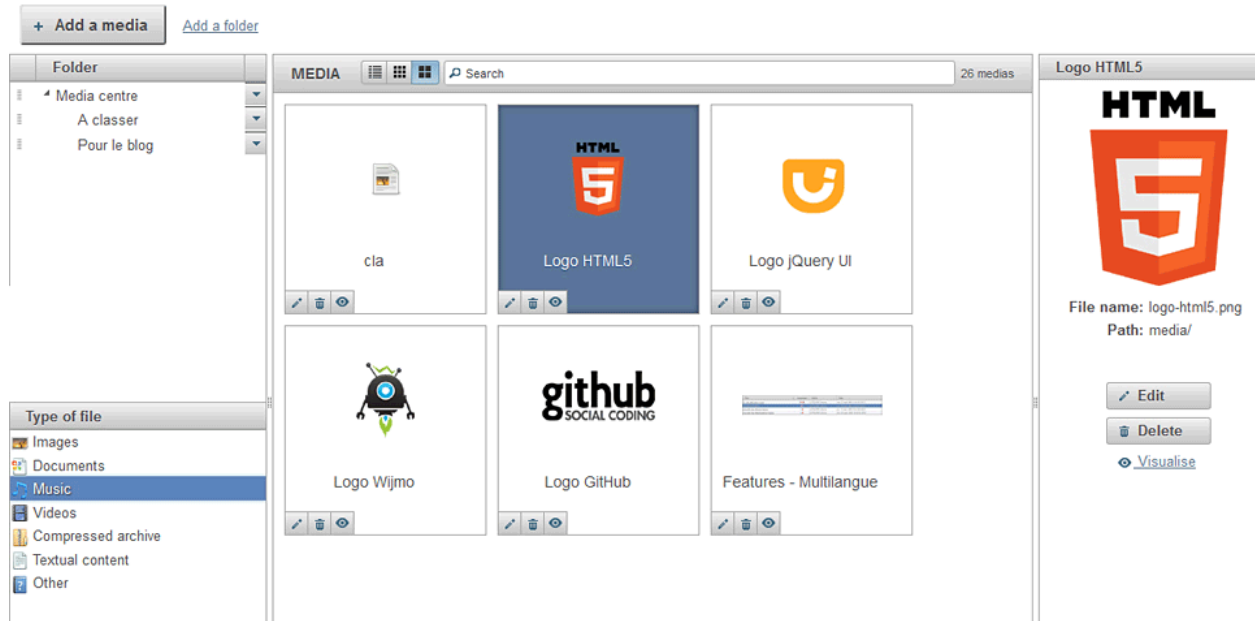
pop-ups must be limited to modal use, i.e, when an action must absolutely be accomplished (or canceled) before carrying out the work (such as confirming a suppression, adding a link or an image to a WYSIWYG content).

The App Desk

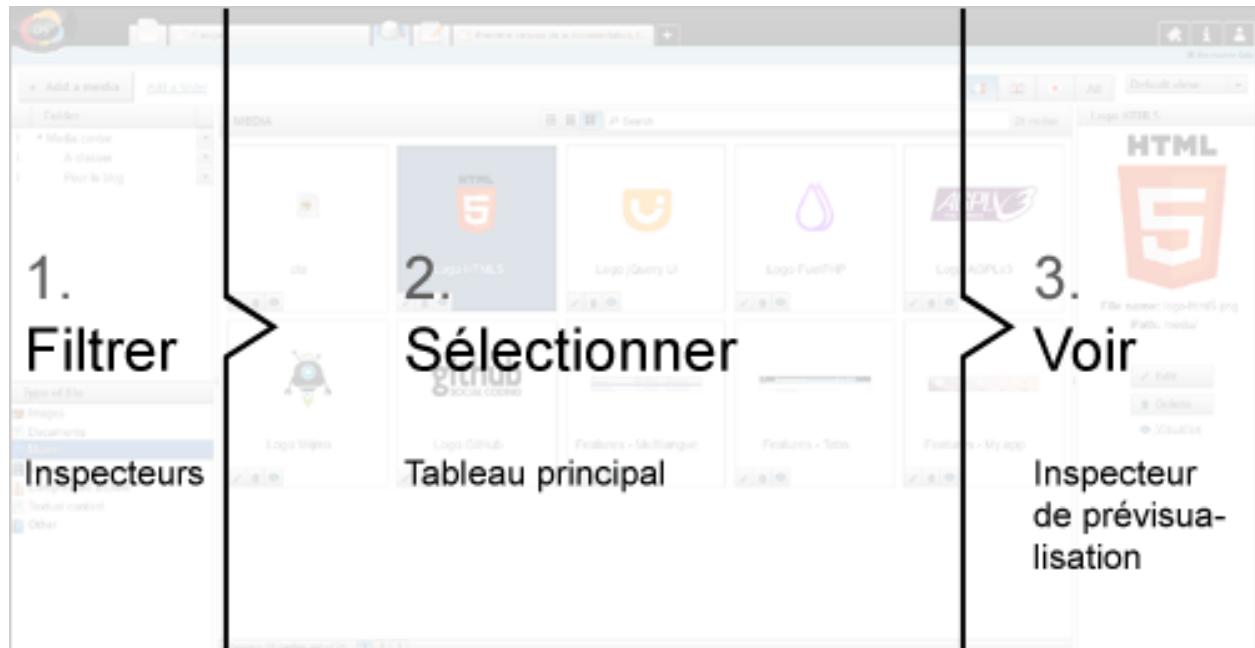
[See the screencast related to the App Desk](#)

The App Desk is the home page of an application, it allows to browse and access the items. It's made of the following elements:

- **Main grid**: list the items of an application, one or several views can be chosen (thumbnails, grid, tree, etc.). Its content is filtered by the inspectors and / or a full-text search. There's only one main grid for each App Desk.
- **Inspectors**: they constitute the meta items of an application (such as blog authors, or media folders). Inspectors allows to filter the content displayed in the main grid (to show the posts of a particular user). Some inspectors allows to handle the data too (for instance, deleting a folder).
 - **Preview inspector**: the preview inspector is a special case. Unlike others inspectors, it doesn't act on the main grid, but rather the main grid acts on it: when an item is selected, details are shown in the preview inspector (image, properties, summary of the possible actions on the item).
- **Actions**: in the vast majority of times, each App Desk must offer one and only one main action, which is *Add a new item*. Secondary actions can also appear as links: adding a meta item (like a folder) or other usual actions (such as exporting data).



The App Desk offers a lot of possible page layouts for developers and end-users. Meanwhile, we recommend to show a [Three-Pane Interface](#) as default.



Alternatively, the preview inspector can be placed under the main grid.

1.2.5 Applications' fundamentals

An application is defined by its models, controllers and views. They vary upon the type of the application, but some principles doesn't change and can be reused for every application.

Defining an application

For an application to be added (installed) using the application manager, it needs a `metadata.config.php` file. This file must contain the application's namespace, written as `Provider\AppName`, along its name, version and provider (at least a name).

An application should also defines one of the following in the `metadata.config.php` file:

Launchers Icon on the home tab, used to launch an application.

Enhancers They allow applications to enhance a WYSIWYG edited content.

Templates Layout for the front-office.

Data catchers Component which allows an application to exploit shared data.

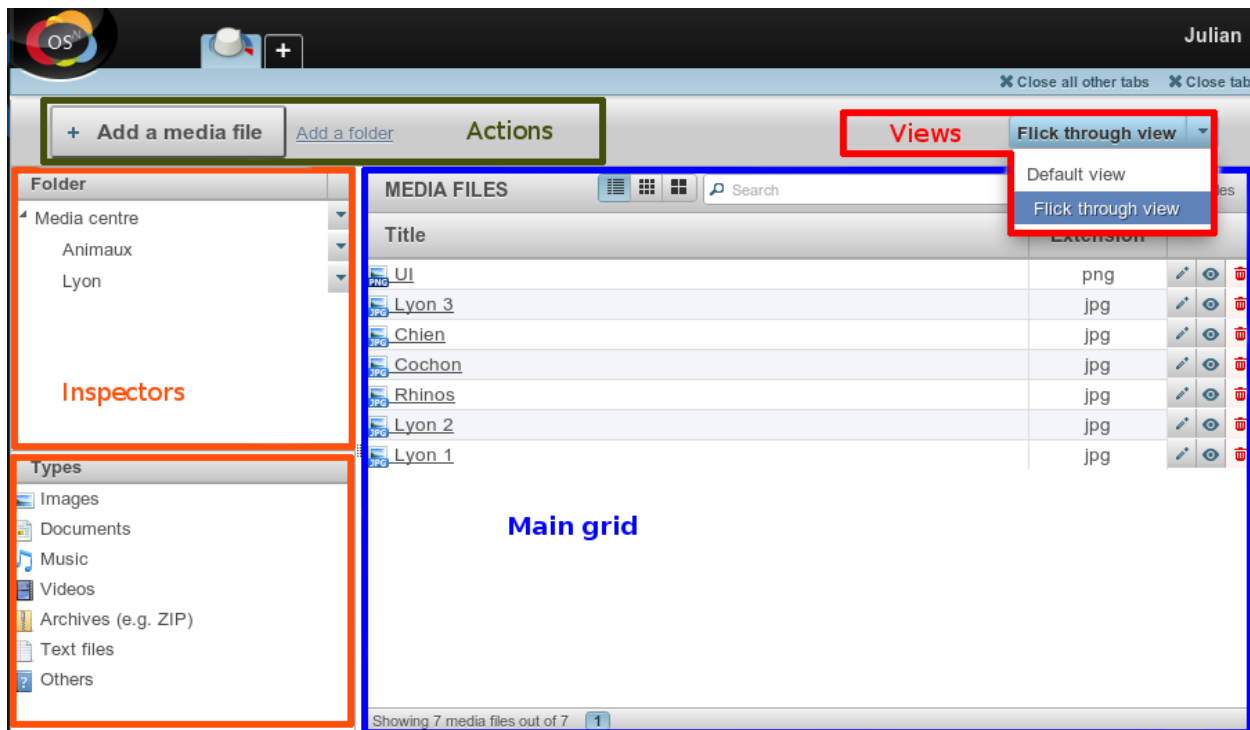
See Also:

'Understanding the applications' infographic

L'App Desk

To understand the App Desk, please *read the ergonomic guidelines* first.

App Desk's configuration



The main feature of the AppDesk is several configurable elements:

- how to display the data ;
- the data itself ;
- main and secondary actions.

The *main grid* can offer several *views* (don't mistake them with the V of MVC) : grid, tree or thumbnails.

These views are defined with configuration files, which specify exactly which data to display and the *actions*.

Inspecteurs are also defined with configuration files. They tell on which attribute or relation the data of the main grid will be filtered, and the actions associated to the meta items contained in the inspector. Please note the inspectors either based on:

- The same model as the one of the main grid (like a publication date), the inspectors then uses a particular column / attribute of the item to filter the results;
- Another model (like posts' author), the inspectors then uses a relation fo filter the results.

Controllers, forms and models

On the App Desk screen, the modifications on the data are operated by calling a controller.

Some operations are made directly (for instance to delete an item, only a confirmation is asked). In this case, the the App Desk's controller is responsible to carry them out.

Other operations needs a specific view and it's the controller of the item which is responsible. Most of the time, the view is a form (new item / edition), which is built using a configuration file and is populated using an instance of the associated Model. The controller is invoked again to save the data when submitting the form.

Observers and behaviours

Observers exists in the [FuelPHP framework](#).

They contain the logic which is directly dependent of the model. They're expressed when a specific event is triggered. They are used to format, change or validate the properties of the model (for instance before adding a new item in the database).

Behaviours are specific to Novius OS, they embrace and extend this principle. Observers can only make specific actions when the model triggers a specific event (such as `before_save`). Behaviours can do more and contain a set of additional methods which defines a particular behaviour, for example translatable or publishable. They also can be triggered by events.

The main advantage of this tools is to share code between different models (horizontal reuse).

See Also:

[API documentation](#)

1.2.6 Multi-Contexts

Multi-contextes basics

Novius OS natively manager several websites, and each of them can have several linguistic versions.

A context is a site / language pair.

Example

Your Novius OS instance can manage your showcase website, which exists in 3 languages (French, English and Spanish), your mobile website, which only exists in French and your events website, which exists in English.

Site / Language	French	English	Spanish
Showcase	X	X	X
Mobile	X		
Events	X		

In this situation, your Novius OS instance manages **5** contexts:

- Showcase / French
- Showcase / English
- Showcase / Spanish
- Mobile / French
- Events / English

Configuration

Everything to configure the contexts is described in the *API documentation*

Cas particuliers

He who can do more can do less. Your Novius OS can manage:

- a single website with several languages;
- several websites in a single language;
- a single website in a single language.

The back-office interface reacts to any of these situations. The `_context_` word will disappear in favor of `_language_` or `_site_`.

It can even vanish completely for a single website in a single language.

Adding contexts

It can be done at any time! Novius OS will happily manage every new contexts, sites or languages found in the configuration. Just change the `contexts.config.php` file and new contexts will be taken into account straight away.

See Also:

API documentation on multi-contexts.

Contextable / Twinnable

Although Novius OS natively manages multiple contexts, each application decides the way it uses them on its own.

Three cases can be found.

Application not using any context

It's the most simple and default case. The application doesn't manage any context and has nothing to do.

Its content will be the same across all contexts and can be used (through *enhancers*) by any of them.

Contextable application

The application manages contexts. Each item will be associated with a context, and should only be used inside of it.

Technically, tables of the application have a `context` column (of type `varchar(25)`) which contains the context code. *Models* of the application must implement the *Contextable* behaviour.

Twinnable application

The application manages contexts and can link them together. Each item is associated with a context and can be linked to other items from a different context.

Technically, tables of the application have 3 columns:

context `varchar(25)`, contains the context code of the item.

common_id_property `int` contains a common ID share among all linked items.

is_main_property `boolean`, each group of linked items will only have one main item.

Models of the application must implement the *Twinnable* behaviour.

Example Schema of our example table:

- `item_id` (primary key)
- `item_context`
- `item_common_id_property`
- `item_is_main_property`
- `item_title`

Let's create a first item:

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main::fr_FR	1	1	Premier item

The `item_common_id_property` column is assigned with the same value as the primary key.

The item is primary, and the `item_is_main_property` is set to 1.

Let's add another item in another context, and linked to the first one:

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main::fr_FR	1	1	Premier item
2	main::en_GB	1	0	First item

The `item_common_id_property` column is assigned with the same `item_common_id_property` to which it's linked to.

`item_is_main_property` is set to 0, it's not the primary item.

Let's see how it looks after a few more addition:

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main::fr_FR	1	1	Premier item
2	main::en_GB	1	0	First item
3	main::en_GB	3	1	Second item
4	main::fr_FR	3	0	Second item (fr)
5	event::fr_FR	5	1	Item du site event
6	main::es_ES	1	0	First item (es)

The items with ID 1, 2 and 6 are linked together, and the main item is 1 / `main::en_GB`.

The items with ID 3 and 4 are linked together, and the main item is 3 / `main::en_GB`.

Let's delete the item with ID 1 :

item_id	item_context	item_common_id_property	item_is_main_property	item_title
2	main::en_GB	1	1	First item
3	main::en_GB	3	1	Second item
4	main::fr_FR	3	0	Second item
5	event::fr_FR	5	1	Item du site vent
6	main::es_ES	1	0	First item

The item with ID 2 now becomes the main item, but the `item_common_id_property` has **not** changed.

1.2.7 Media centre

General informations

The media centre is the central place which gather most of the files used by the applications. It contains images, documents, videos or any other file.

- All files are store in the **private** directory `/novius-os/local/data/media/`
- They are accessed using the URL `http://your.website.com/media/folder/ressource.ext`

Functioning

When accessing a media for the first time, the 404 handle is invoked. It creates a symbolic link in the `public/media` directory, so subsequent requests don't need to use the 404 handler anymore.

It works this way because we'll be able to handle **private** medias in the future. The latter will return:

- a HTTP 401 error code (authorization required);
- or the file will be sent on the standard output, but without creating a symbolic link (permissions need to be checked for each and every request).

Optimisation

When PHP sends a big file on the output, it blocks the process until the transfer is completely done. But it's possible to release the process instantaneously by delegating the work to the underlying web server (usually Apache or nginx).

The `XSendfile` mechanism can be used. It consists of sending a special header from the PHP script, its name can vary upon one server or another:

- `X-Sendfile` is used by **Apache** and some others ;
- `X-Accel-Redirect` is used by **nginx**.

See Also:

Post-install optimisations

Attached files (outside the media centre)

You may not want to store all your files in the media centre. For instance, a 'human resources' application collects CV of candidates, and you don't want them to be visible in the media centre.

There's an *associated file* mechanism to handle this case. It works roughly the same as email attachments and allows to store a CV with its candidates data.

1.2.8 Permissions

Roles (or "profiles")

Permissions are always applied to a role. Then, each user is given one or many roles, from which the permissions are determined.

One role by user

For the sake of simplicity and understanding, on a default installation:

- these roles are hidden ;
- each user gets one unique role attributed ;
- it's not possible to share a role among several users.

Hence the role notion is completely hidden and the permissions are configured on a dedicated tab of the user form :

The screenshot shows a web interface for user management. On the left is a sidebar with two tabs: 'User details' and 'Permissions'. The 'Permissions' tab is selected and highlighted with a red rectangle. The main content area is divided into two sections. The top section, titled 'Details', shows the user's name 'Julian' and a role 'ESPERAT'. Below this, it displays the email address 'esperat@novius.com', the last signed in time '10:50, 03 April 2013', and the language 'English'. There is also a checkbox for 'Expert view' which is checked. The bottom section, titled 'Set a new password', contains two input fields for 'Password' and 'Password (confirmation)'.

Several roles by user

Meanwhile it's possible to enable multiple roles in the main configuration file :
`novius-os.users.enable_roles = true.`

Once enable, it becomes possible to share a role among several users. Permissions can then be configured more precisely:

- on the user form, the “Permissions” tab disappear in favor of a “Roles” block ;
- the AppDesk of the Users application gains:
 - a new “Roles” inspector ;
 - a new “Add a role” action.
- permissions should now be configured on the role form.

Julian

ESPERAT

Details

Email address: *

esperat@novius.com

Last signed in on:

10:50, 03 April 2013

Language:

English

☒ Expert view

Roles

☒ This is a role

Set a new password

Password:

Repeat (confirmation):

Permission structure

They are two types of permission:

- simple: yes or no ;
- multiples : applied on a list of categories (values);

A **simple** permission has a meaning just by itself. For instance “Can I add a page?” or “Can I delete a locked page?”.

A **multiple** permissions don’t have a meaning just by themselves, and can only be expressed depending on a category (value). For instance with “Can I write in this folder?” we need a list of folders for the permission to have a meaning. With “Can I access this application?” we need a list of applications.

A **simple** permission is composed of a unique `perm_name` column, whereas a **multiple** role (using categories) is composed of two columns: `perm_name` (the same on as a simple permission) and `perm_category_key` (the value).

How-to use permissions in the applications





`permissions.config.php` file

By creating this file, each application can define its own list of permissions it needs.

They can be configured in the dedicated column when editing the permissions :

Can access the following applications:

☐ Check all

<input checked="" type="checkbox"/>	 Sample application	<div>Permissions for this application</div> <div><input checked="" type="checkbox"/> Can create new items</div> <div><input type="checkbox"/> Can delete locked items</div>
<input checked="" type="checkbox"/>	 Media Centre	
<input checked="" type="checkbox"/>	 Webpages	
<input checked="" type="checkbox"/>	 Users	

See Also:

associated API for the permission configuration file

API to check a permission

```
<?php
// Simple: only 1 argument, the permission name
Permission::check('noviusos_app::delete_locked');

// Multiple (uses categories) : 2 arguments, the permission name + the category key
Permission::check('noviusos_app::create_in_folder', $folder_id);
```

See Also:

API documentation of the *Permission* class.

Warning: The permission name is an important thing. The part preceding `::` must refer to an valid application. For the permission to be granted, the user also needs to have access to this application.

CRUD

It's possible to hide some fields based on the permissions. To do so, the *show_when* key defines a callback function returning whether the field should be visible or not.

```
<?php
return array(
    'fields' => array(
        'my_field' => array(
            'label' => 'My field',
            'form' => array(
                'type' => 'text',
            ),
        ),
        'show_when' => function() {
            // The field will only be visible when the user has the requested permission
            return Permission::check('my_app::my_permission');
        },
    ),
);
```

```

    ),
  ),
);

```

Actions

It's possible to disabled actions based on the permissions using the *disabled* key.

```

<?php
return array(
  'data_mapping' => array(/...*/),
  'actions' => array(
    'delete' => array(
      'label' => __('Delete'),
      'primary' => false,
      'icon' => 'home',
      'action' => array(/...*/),
      'targets' => array(
        'grid' => true,
      ),
      'disabled' => array(
        function($item) {
          return !Permission::check('my_app::can_delete_item') ? __('You don\'t have the p
        )
      ),
    ),
  ),
);

```

1.2.9 Front-Office and cache

Novius OS is using Apache **mod_rewrite** (or any equivalent on other servers) to display pages in the front-office.

Every URL ending with `.html`, the home page and the folders are redirected to the `NOSROOT/public/htdocs/novius-os/front.php` file.

This file loads Novius OS and asks the *Controller_Front* to handle the URL.

The *Controller_Front* parses the URL and figure out the associated cache file path. From there, several things can happen.

Cache execution

When the cache file associated to the current URL exists.

The cache is saved in the `NOSROOT/local/cache/page/` directory. First level of the tree is the domain name. Below, the URL path is used.

The cache file is a PHP file. First lines are responsible to check whether the cache file is still valid or it has expired. It also has a way to recreate the *Controller_Front* properties which were available when the cache was generated (page instance, URLs, status, headers, custom data).

If it's still valid, the cache is executed and display the page to the user, with the saved `status` and `headers`.

Cache generation

When the cache file associated to the current URL doesn't exist or has expired.

The *Controller_Front* will find the appropriate page based on the URL. Once the page is known, it can determine the associated template and the WYSIWYG list to insert into the template.

When WYSIWYG contains enhancer, they are executed and their content is saved.

Then the template (it's a View) is executed with the following data: \$wysiwyg array, the page \$title, the \$page and the *Controller_Front* (\$main_controller).

The generated content is saved into the cache file.

Once the cache is generated, it's being executed (see above section) to send the content to the user, with the status and headers specified during the cache generation (especially by the enhancers).

Possible interactions

During the process *Controller_Front* triggers several events at key locations. By using them, it's possible to alter the process.

See Also:

Front-office events

You also have control on the process by using the *Controller_Front* methods. You can retrieve the front controller instance using *NosNos::main_controller()*, or just use \$this->main_controller inside an enhancer.

Alter the generated content

In some situations, you may want to generate your own content and skip the page template. For instance, an enhancer can send a RSS feed. It's done using the `sendContent()` method of the *Controller_Front*.

Below is an example (enhancer code):

```
<?php

$this->main_controller->setHeader('Content-Type', 'application/xml');
$this->main_controller->setCacheDuration(60 * 30); // Cache duration is set to 30min
return $this->main_controller->sendContent($rss); // The $rss variable contains the RSS feed (XML content)
```

The cache file will only contain the RSS feed content and the HTTP response will contain a header with the correct content-type.

Executing outside the cache

In some situations, the caching system is too much effective. For instance, if a portion of the template or of the enhancer should be different depending on whether the user is logged in or not. In this situation, it's useful to tell the cache to execute a PHP code each time rather than saving its result.

To do so, the *FrontCache* provides the `callHmvcUncached()` and `viewForgeUncached` methods.

```
<?php

// This will execute a controller's action each time the cache is executed.
\nos\FrontCache::callHmvcUncached(
    'uri/controller',
```



```

        array(
            'id' => \My_User::get_current_user_id() // Just as an example, the My_User class doesn't rea
        )
    );

// or

// This will include the view each time the cache is executed (rather than saving its result)
\nos\FrontCache::viewForgeUncached(
    'uri/view', // View path
    array(
        'id' => \My_User::get_current_user_id()
    ),
    false
);

```

Suffix Handler

You can configure the cache path to also vary based upon any parameter, in addition to the current URL. For instance, different GET parameters doesn't change the cache path (the same file is used for the same URL, with or without GET).

To do so, use the `addCacheSuffixHandler()` method from the *Controller_Front*.

```

<?php

\nos\nos::main_controller->addCacheSuffixHandler(array(
    array(
        'type' => 'GET',
        'keys' => array('my_param'),
    ),
));

// or

// The callback function must return a string (empty string when you don't want to alter the cache p
\nos\nos::main_controller->addCacheSuffixHandler(array(
    array(
        'type' => 'callable',
        'callable' => array('MyClasse', 'myMethod'),
        'args' => array(
            'example arg'
        ),
    ),
));

```

In the first example, the cache system will generate one cache files for each different value of the `GET[my_param]` variable.

In the second example, the cache system will call the `MyClasse::myMethod('example arg')` method, which is responsible to return a suffix to the file path if necessary.

1.3 Create a new application

1.3.1 The application wizard

The application wizard allows you to easily create a new application : Models, Fields and Field groups, App Desk, Launchers, URL enhancers...

The application wizard allows you to skip repetitive tasks and focus on what is important.

Warning: The last step of the application wizard is table and files generation. This files will be in a new folder (which name you have chosen) is `local/application/`. Therefore, Novius OS (user `Apache` if Novius OS runs on a **Apache**) has to have write rights on this folder.

1.3.2 Add an enhancer

1. Configuration in `metadata` file

Metadata of an enhancer are described in the *API documentation*.

2. [Back-office] Create the enhancer's controller

In order to manage the configuration popup as well as the enhancer's preview, we need a controller.

Create the file `my_app::classes/controller/admin/enhancer.ctrl.php` extending `Controller_Admin_Enhancer`.

```
<?php
```

```
namespace My\App;
```

```
class Controller_Admin_Enhancer extends \Nos\Controller_Admin_Enhancer
{
}
}
```

As most Novius OS controllers, we can add a configuration file: `my_app::config/controller/admin/enhancer.config.php`.

```
<?php
```

```
return array(
    // Empty, for the moment
);
```

Configuration popup

Popup display depends on the `dialog` key in the `metadata.config.php` file.

```
<?php
```

```
return array(
    'enhancers' => array(
        'my_app' => array(
            'dialog' => array(
```

```

        'contentUrl' => 'admin/my_app/enhancer/popup',
        'ajax' => true,
    ),
),
);

```

Here, the popup will call the function `action_popup()` of the `Mon\Appli\Controller_Admin_Enhancer` class using ajax.

As any change on a `metadata.config.php` file, you need to apply changes in the application manager.

From now on, when we add an enhancer into a WYSIWYG, a popup appears, but the configuration form is empty.

Standard controller we extended expects a configuration in order to add options into the popup. In the `mon_appli::config/controller/admin/enhancer.config.php` file:

```

<?php

return array(
    // Popup configuration
    'popup' => array(
        'layout' => array(
            'view' => 'my_app::enhancer/popup',
        ),
    ),
);

```

The file refers to the `mon_appli::enhancer/popup` view which doesn't exist, we need therefore to create. It will contain form inputs managing configuration:

```
<h3>Options</h3>
```

```
<p>
```

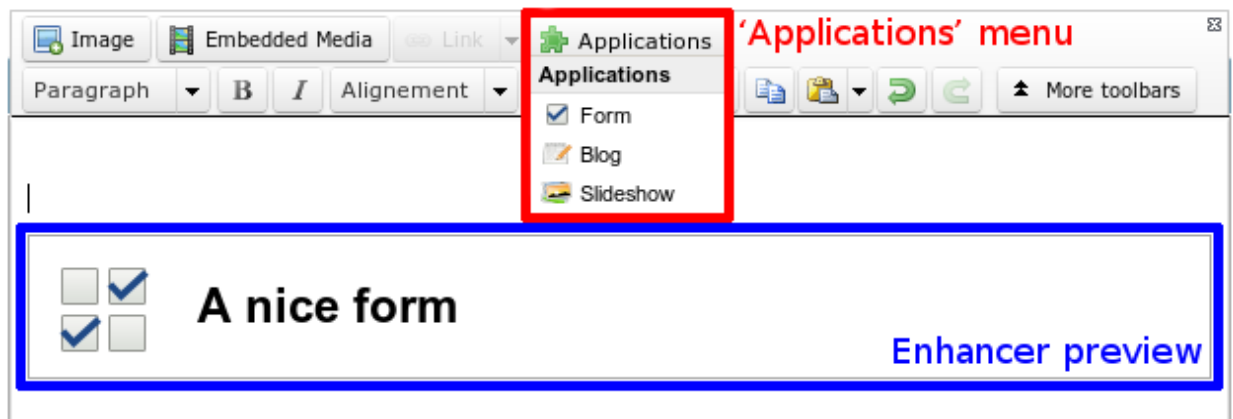
```
    <label for="item_per_page"><?= __('Item per page:') ?></label>
```

```
    <input type="text" name="item_per_page" id="item_per_page" value="<?= \Arr::get($enhancer_args,
```

```
</p>
```

Old enhancer configuration can be accessed with the `$enhancer_args` variable (useful for pre-fill form with existing configuration).

Change preview



The preview added into the WYSIWYG is loaded by calling `previewUrl` key in the `metadata.config.php` configuration file.

Generally, the same controller than the popup is called, only the action is changed to `action_preview()`.

View provided by default uses an icon, a title (default value is the 64x64 application icon, and the enhancer title), as well as a layout (additional view files called).

`my_application::config/controller/admin/enhancer.config.php`:

```
<?php

return array(
    // Popup configuration
    'popup' => array(
        // Already dealt with in previous part
    ),
    // Preview configuration
    'preview' => array(
        // (optional) view to be used in order to render (default value is written beneath):
        //'view' => 'nos::admin/enhancer/preview',
        // (optional) additional view files (included par la view au-dessus)
        //'layout' => array(),
        'params' => array(
            // (optional) default value is enhancer title
            'title' => "Mon super enhancer",
            // 'icon' (optional) default value is application 64x64 icon
        ),
    ),
);
```

Value can be callback functions for title and icon. This callback will receive only one parameter: the enhancer configuration.

One example is the « Form » enhancer where the selected form title is also displayed.

3. [Front-office] Display content on the website

Once the page has been save and published, the enhancer will appear on the website.

The content will be generated by the controller defined on keys `enhancer` or `urlEnhancer` of the `metadata.config.php` file (whether we wanted a simple or URL enhancer). Don't forget to apply changes in the application manager if you change `metadata.config.php`.

For instance, the value of the `enhancer` key for the « Form » application is `noviusos_form/front/main`, so it will call the `action_main()` method of `Controller_Front` of the `noviusos_form` application (`Nos\Form\Controller_Front` class).

The first parameter of this action is the configuration table defined by user on the configuration popup.

Create a controller in `my_app::controller/front.ctrl.php`

```
<?php

namespace My\App;

class Controller_Front extends \Nos\Controller_Front_Application
{
    public function action_main($enhancer_args = array())
    {
```

```

        // Testing
        return print_r($enhancer_args, true);
    }
}

```

4. URL enhancers

URL enhancer are capable of managing URLs.

Concretely, if your URL enhancer has been added to the my/page page, then it will be able to manage urls begining with my/page/**.html, as:

- my/page.html
- my/page/first_level.html
- or my/page/first_level/second_level.html

There is no limitation on the level you can manage.

As on previous part, content is generated by the main action, but it is possible to get the extended url with `$this->main_controller->getEnhancerUrl()` ;.

The controller can therefore switch content depending the called URL. Here is an (simplified) example taken from the « Blog » application :

```

<?php

namespace Nos\Blog;

class Controller_Front extends \Nos\Controller_Front_Application
{
    public function action_main($enhancer_args = array())
    {
        // Complete url == 'my/blog/category/ski.html'
        // => $enhancer_url == 'category/ski' (without .html)
        $enhancer_url = $this->main_controller->getEnhancerUrl();
        $segments = explode('/', $enhancer_url);

        if (empty($enhancer_url))
        {
            // URL is 'mon/blog.html' (page URL)
            // Display list of blog posts (first page)
        }
        else if (count($segments) == 1)
        {
            // URL is 'mon/blog/blog_title.html'
            // Blog post 'blog_title' is displayed
        }
        else if (count($segments) == 2)
        {
            if ($segments[0] == 'page')
            {
                // URL is 'my/blog/page/number.html'
                $page = $segments[1];
                // Display page number of blog posts list
            }
            else if ($segments[0] == 'category')
            {

```

```
        // URL is 'my/blog/category/category_name.html'
        $category = $segments[1];
        // Display blog posts list of 'ski' category
    }
}

// URL called isn't managed by the enhancer (404 error)
throw new \Nos\NotFoundException();
}
}
```

When an enhancer manage URLs for some models (ORM), it must know the mapping between models and URLs. This is allowed by the static method `get_url_model()`:

```
<?php

namespace Nos\Blog;

class Controller_Front extends \Nos\Controller_Front_Application
{
    public static function get_url_model($item, $params = array())
    {
        $model = get_class($item);
        $page = isset($params['page']) ? $params['page'] : 1;

        switch ($model)
        {
            // Blog post URL
            case 'Nos\Blog\Model_Post' :
                return urlencode($item->virtual_name()).'.html';
                break;

            // Category URL
            case 'Nos\Blog\Model_Category' :
                return 'category/'.urlencode($item->virtual_name()).($page > 1 ? '/'.$page : '').'.html';
                break;
        }

        return false;
    }
}
```

This function is related to the `Behaviour_URLEnhancer` and `url()` and `urls()` model methods. In order to understand how to configure them, take a look at the [API documentation](#).

Example :

```
<?php

// Selecting the category which ID value is 1
$category = Nos\Blog\Model_Category::find(1);

$url = $category->url(array('page' => 2));
```

\$url value will be `my/blog/category/ski/2.html`:

- `my/blog` : Page URL;
- `ski` : Category virtual url ;

- 2 : Page number.

1.3.3 Create a template

1. metadata configuration

Template metadata are described in *the API documentation*.

2. View file creation

File location depends on the `file` key configured in the `metadata.config.php` file.

Inside the template, some variable can be accessed:

\$wysiwyg A hash which keys are the WYSIWYG name configured in the `metadata.config.php` file and values are content the user entered.

\$page `Nos\model_Page` instance.

\$main_controller *Front controller instance*.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <link rel="shortcut icon" href="static/favicon.ico" />
  <link rel="stylesheet" type="text/css" href="static/apps/noviusos_templates_basic/css/base.css" />
</head>

<body>

  <header>This header will be displayed on all pages configured to use this template.</header>

  <div id="menu">
  </div>

  <div id="content">
    <?= $wysiwyg['content']; ?>
  </div>

</body>
</html>
```

1.3.4 Add fields

See Also:

`/app_extend/add_field`

Most fields added need a column in the model associated MySQL table.

Fields are then added in the CRUD form using the `fields` key in the configuration file.

Syntax used is using a existing feature, which defines how a column displays.

See Also:

[FuelPHP documentation about model properties](#)

Moreover, Novius OS team implemented *renderers*, which allows more freedom. Some renderer allow to select medias, pages, date.

Configuration example:

```
<?php
return array(
    'name' => array(
        'label' => 'Text displayed next to field',
        'form' => array(
            'type' => 'text',
            'value' => 'Default field',
        ),
        'validation' => array(),
    );
```

Standards fields

Bold text is the type property value:

- `<input type="text">`
- `<input type="password">`
- `<textarea>`
- `<select>`
- `<input type="radio">`
- `<input type="checkbox">`
- `<input type="submit">`
- `<input type="button">`
- `<input type="file">`

```
<?php
return array(
    'gender' => array(
        'label' => 'Gender',
        'form' => array(
            'type' => 'select',
            'options' => array(
                'm' => 'Male',
                'f' => 'Female',
            )
        ),
        'validation' => array('required'),
    ),
);
```

`<button type="submit">`

- type = submit generate `<input type="submit">`
- type = button generate `<input type="button">`

tag property can be used to force HTML tag, for the submit button case.

FuelPHP use automatically value as button text.

```
<?php
return array(
    'save' => array(
        'form' => array(
            'type' => 'submit',
            'tag' => 'button',
            'value' => 'Save',
        ),
    ),
);
```

Renderers (enhanced fields)

renderers list is available in *API documentation*.

1.3.5 Add thumbnails view in App Desk

It is actually quite simple. You need to define two special keys in data_mapping:

- thumbnail: thumbnail item path ;
- thumbnailAlternate : default path when no item thumbnail path is defined.

In the file config/common/item.config.php:

```
<?php
return array(
    'data_mapping' => array(
        'thumbnail' => array(
            'value' => function ($item) {
                foreach ($item->medias as $media) {
                    return $media->get_public_path_resized(64, 64);
                }
                return false;
            },
        ),
        'thumbnailAlternate' => array(
            'value' => function ($item) {
                return 'static/apps/mon_appli/icons/64.png';
            },
        ),
    ),
);
```

You need then to enable the thumbnails view in the App Desk configuration my_app::config/controller/admin/appdesk.config.php:

```
<?php
return array(
    'model' => '',
    'query' => array(),
    'inspectors' => array(),
```

```
'il8n' => array(),
'thumbnails' => true,
);
```

If you want to show the thumbnails view by default:

```
<?php

return array(
    'model' => '',
    'query' => array(),
    'inspectors' => array(),
    'il8n' => array(),
    'thumbnails' => true,
    'appdesk' => array(
        'appdesk' => array(
            'defaultView' => 'thumbnails',
        ),
    ),
);
```

1.3.6 Add attachment

Principles

The class *Attachment* allows you to manage attachments.

Attachments are saved in the `local/data/files/` directory.

Usually attachments are joined to a *Model* but it is possible to manage them as you wish. You just need a tiny configuration in order to define an *Attachment*.

```
<?php

$attachment = \Nos\Attachment::forge('my_id', array(
    'dir' => 'apps'.DS.'myapps',
));
```

On above example, attachment will be saved in the `local/data/files/apps/myapps/my_id/` directory.

In order to save a file, you will just need:

```
$attachment->set($_FILES['file']['tmp_name'], $_FILES['file']['name']);
$attachment->save();
```

On above example, we save an uploaded file as an attachment. File location will be: `local/data/files/apps/myapps/my_id/original_name.ext` where `original_name.ext` is the original name of the uploaded file collected from `$_FILES['file']['name']`.

Joined to a model

In the case a file is joined to a *Model*, it is even simpler. In your class, you just need to write:

```
class Model_Example extends \Nos\Orm\Model
{
    protected static $_attachment = array(
```

```
        'avatar' => array(),
        'document' => array(),
    );
```

This way, each `Model_Example` item will have two attachment: `avatar` and `document`.

```
$item = Model_Example::find('first');
$item->avatar->set($_FILES['file']['tmp_name'], $_FILES['file']['name']);
$item->avatar->save();
```

Details

For more details, check [Attachment](#).

Extensions

When you create a new [Attachment](#), you can specify a list of authorized extensions by adding the `extensions` key to the configuration. Value should be an array of authorized extensions.

If your file has to be an image, you can set the special key `image` to `true`.

URL alias

By default, your attachment will be available in this URL:

```
http://www.domain.com/data/files/dir/id/file_name.extension
```

If `dir` value is, as often, `apps/my_app/my_file_type/`, the url can be quite extensive.

Define an `alias` class in your [Attachment](#) configuration. `alias` value will replace `dir` value in URL.

Secured attachment

It is possible to secure your attachments, in order to limit access only for authenticated user for example. You just need to define on configuration the `check` key which value is a [fonction de callback](#). Each time file is requested, the system will execute this function, with first parameter the current [Attachment](#) instance, in order to check the file is available in this context.

Example:

```
class Verification
{
    public static function check($attachment)
    {
        return isset($_SESSION['user_connected']) && $_SESSION['user_connected'];
    }
}

$attachment = \Nos\Attachment::forge('my_id', array(
    'dir' => 'apps'.DS.'myapps',
    'check' => array('Verification', 'check'),
));
```

In the upper example, if the user is logged (session key `user_connected` set to `true`), the file will be available. If not, the url will throw a 404 error.

1.3.7 Translate an application

Each application possess the `lang` directory where are located translation files, that will be called **dictionaries**.

This directory include as many subdirectories as languages we want to translate the application into. Dictionaries will be define in these subdirectories.

These language directories are named following locales, as for example `fr` (French), or `en` (English).

Dictionaries are PHP files returning a array, similarly as configuration file.

See Also:

I18n class API

File `metadata.config.php`

`metadata` are a particular case, since they are cached. They need a translation file on their own. First step is to define which dictionaries `metadata` need to be translated.

```
<?php

return array(
    'name' => 'My app',
    'namespace' => 'My\App',
    'i18n_file' => 'my_app::metadata',
    // ... other keys
);
```

As all change on `metadata`, don't forget to apply changes in the application manager.

Next, you need to create the `my_app::lang/fr/metadata.lang.php` dictionary:

```
<?php

return array(
    'My app' => 'My application',
);
```

Novius OS automatically knows which keys has to be translated in the `metadata` file and will get corresponding translations.

Other files

Elsewhere, you need to do the following in order to translate files (including PHP):

1. Use the `__()` function in order to get translations;
2. For each file, configure in which dictionary will be used translations.

It is quite simple for view and configuration files:

```
<?php

// Configure the __() function
Nos\I18n::current_dictionary('my_app::common');

__('Translate this'); // Translation will be collected from my_app::lang/<lang>/common.lang.php
```

It is a little more complicated for admin controllers, because language depends on the user and is known only after authentication, which happens in `before()`.

`prepare_i18n()` has been implemented to solve this problem:

```
<?php

namespace Nos\Form;

class Controller_Admin_Form extends \Nos\Controller_Admin_Crud
{
    public function prepare_i18n()
    {
        // Configure language file depending on user
        parent::prepare_i18n();
        // Configure the __() function
        \Nos\I18n::current_dictionary('noviusos_form::common');
    }

    // Other methods using __()
}
```

It is possible to use many dictionaries in only one file ; just use an array instead of a string. Translation will be choose from the first file containing required key.

```
<?php

Nos\I18n::current_dictionary(array('my_app::dictionary', 'my_app::common'));

// Translation will be collected from my_app::lang/<lang>/dictionary.lang.php if it exists
// Otherwise in my_app::lang/<lang>/common.lang.php
__('Translate this');
```

1.3.8 Migrations files

Each application as well as the local folder can have a *migrations* folder. This folder contains migrations files that are a convenient way to update the database or files. The [FuelPHP migration system](#) is used. However on Novius OS there are two things you should know:

- Application migration files must be on the namespace `{{APPLICATION_NAMESPACE}}\Migrations`
- Whenever it is possible, sql update must be on a separate file (in order to make easier manual updates). As most of the time only sql requests are executed, a migration class has been implemented in order to ease migrations. You can take a look at the [API documentation](#).

When an application is installed or updated, migration files are executed (if they haven't been already) through the application manager.

1.4 Extend an application

1.4.1 Extensions mechanisms

Create a file in `local`

Any view or configuration file can be changed via the `local` folder.

This is possible thanks to the cascading file system existing in FuelPHP and adapted for Novius OS. It's very easy to do, because you only need to copy an existing file and change it how you like!

For an application, the file should be copied into `local/config/apps/{application}/` or `local/views/apps/{application}/`.

To extend a file from the core of Novius OS, we'll use `local/config/apps/novius-os/` and `local/views/apps/novius-os/`.

The 'generic' pattern is `local/{section}/{application}/` with:

- `{section}` ` equals to config or views ;
- `{application}` ` matching apps + an application name or novius-os for the core.

Configuration

The `noviusos_page` application has a `controller/admin/appdesk.config.php` configuration file (so it's located at `noviusos_page::config/controller/admin/appdesk.config.php`).

If we copy it into `local/config/apps/noviusos_page/controller/admin/appdesk.config.php`, then it will be merged automatically with the one asked by the application.

Views

When we create a `local/views/apps/noviusos_help/admin/help.view.php` file, it will be used as a **replacement** of `noviusos_help::admin/help.view.php`!

To extend a file from the core, we'll use `novius-os` as application name. For example, `local/views/novius-os/admin/login.view.php`.

Use events to alter a configuration

Any configuration file can be altered thanks to the `config|<path>` event.

Replace a view with another one

It's possible to call the `View::redirect()` method to replace any view file by another one.

```
<?php
```

```
// Replace the 'admin/help' view of the 'noviusos_help' application by the 'help' view of the 'local'
View::redirect('noviusos_help::admin/help', 'local::help');
```

Create a dedicated extension application

To extend an application, a dedicated application can be created, which will alter how the first one works.

The second application defines its extending `my_application` through its `metadata.config.php` file:

```
<?php
```

```
return array(
    'name' => 'Application 2',
    // It's an extension application
```

```
'extends' => 'my_application',  
);
```

Once application_2 is installed, it will be loaded at the same time than my_application is.

When an application extends another one, some automatic behaviours falls into place.

Example:

application_2 extends my_application.

Configuration files of Controller and Model inside my_application can automatically be extended by application_2 just by creating them at the same location.

For instance, my_application has the following configuration file for Controller_Test: applications/mon_application/config/controller/test.config.php.

In application_2, if the matching file applications/application_2/config/controller/test.config.php exists, then it will be merged.

i.e. in My\Application\Controller_Test, the \$config variable will contain the merge of the 2 files (the one of the extended my_application application, and also the one from application_2 which extends the first one).

1.4.2 Bootstrap

The bootstrap file allows you to execute php code when the website / an application is loaded. It can be placed in two locations :

- local/bootstrap.php: will be executed when the website is loaded.
- local/applications/APPLICATION/bootstrap.php: will be executed when the application APPLICATION is loaded.

It is possible to use the bootstrap to extend an application. It is there *events* and *view redirects* can be used.

1.4.3 Adding a field

We'll start from an example to explain how it works.

Let's add a Source field on blog posts, to allow us to fill in an external URL from where the original content was produced.

In the database

```
ALTER TABLE `nos_blog_post` ADD `post_source` VARCHAR(255);
```

In the model

2 choices:

- Declare the new field in the model properties.
- Activate the cache mechanism of models properties.

Declare the field

We're going to listen the event on the model config file.

```
<?php
```

```
Event::register_function('config|noviusos_blog::model/post', function(&$config) {
    $config['properties']['post_source'] = array(
        'default' => null,
        'data_type' => 'varchar',
        'null' => false,
    );
});
```

See Also:

Defining properties in FuelPHP documentation

Activate the properties cache

- Create the file `local/config/config.php` by copying `local/config/config.php.sample` (if necessary).
- Uncomment the line (or create it) with the key `cache_model_properties` and set it to `true`:

```
<?php

return array(
    //...

    'novius-os' => array(
        //...
        'cache_model_properties' => true,
        //...
    ),
);
```

When activated, all models properties will be cached in the directory `local/cache/fuelphp/model_properties/`. When a column is added and not declared, the first call to `get()` or `set()` for this column will fetch the schema from the DB and update the cached properties.

Warning: This mechanism only works with the MySQL and MySQLi drivers.

See Also:

Documentation for Novius OS configuration.

In the form

The addition / edition form of a blog post is defined in its CRUD configuration. To extend it, we'll use an event!

In the `local/bootstrap.php` file (create it if necessary):

```
<?php
```

```
Event::register_function('config|noviusos_blog::controller/admin/post', function(&$config) {
```



```
// Add a 'post_source' field (type 'text')
$config['fields']['post_source'] = array(
    'label' => 'Source originale :',
    'form' => array(
        'type' => 'text',
        'placeholder' => 'http://',
    ),
);

// Display the field inside the form
// We create a new 'Source' expander in the right menu
$config['layout']['menu']['Source'] = array('post_source');
});
```

The form now contains an additional editable field, as you can see below:

The screenshot shows the Novius OS interface for creating a new post. The top bar includes the user name 'Julian' and navigation links. The main form area has a title 'Un billet d'exemple', a 'Chapeau' (excerpt) field, and a 'Source' field with the value 'http://www.novius-os.org'. The 'Source' field is highlighted with a red box. The sidebar on the right contains a list of settings: 'Propriétés', 'URL (adresse du billet)', 'Catégories', 'Tags', and 'Source'. The 'Source' setting is expanded, showing the 'Source originale' field with the value 'http://www.novius-os.org'.

In the visualisation

For the view, let's create the `local/views/apps/noviusos_blognews/front/post/content.view.php` file.

```
<?php

// Let's include the original file (it displays the content)
include APPPATH.'/applications/noviusos_blognews/views/front/post/content.view.php';

// And add the 'source' field right after
if (!empty($item->post_source)) {
    ?>
    <p class="blognews_source">
        <?= __('Source:') ?>
    </p>
}
```

```
<a href="<?= htmlspecialchars($item->post_source) ?>">
    <?= htmlspecialchars($item->post_source) ?>
</a>
</p>
<?php
}
```

1.4.4 Changing the appearance on the website

We'll start from an example to explain how it works.

On the [Novius OS](#) website, we personalised how the blog posts are displayed. Here's how it looks like:

Default design of the 'Blog' application :

PHP Tour: FuelPHP conference by the Novius OS team

The PHP Tour is one of the biggest PHP events held in Europe. This year it takes place in Nantes on November 29th and 30th. We will be there!

Author:

16:08, 12 November 2012

Tags: conference, fuelphp

No comments

Contest: five Novius OS T-shirts to win

To celebrate the release of Novius OS wallpaper for Smashing magazine, we give away five "Data catchers VS Content nuggets" T-shirt.

Author:

15:32, 31 October 2012

Tags: content-sharing, goodies, contest

No comments

Forms applications: take our user test

Personalised design on the [Novius OS.org](#) website (our goal):

PHP Tour: FuelPHP conference by the Novius OS team

The PHP Tour is one of the biggest PHP events held in Europe. This year it takes place in Nantes on November 29th and 30th. We will be there!

Created on Monday 12 November 2012 16:08Tags: conference, fuelphp

Contest: five Novius OS T-shirts to win

To celebrate the release of Novius OS wallpaper for Smashing magazine, we give away five "Data catchers VS Content nuggets" T-shirt.

Created on Wednesday 31 October 2012 15:32Tags: content-sharing, goodies, contest

Forms applications: take our user

One of the features to be released with Novius OS 0.2 in December is the Forms application. It will

Changing the view

1st solution: extending the view

Thanks to the cascading file system, we can copy the original `noviusos_blognews::views/front/post/item.view.php` file in our local directory: `local::views/apps/noviusos_blognews/front/post/item.view.php`

```
<div class="blognews_post blognews_post_item">
  <div class="blognews_primary_information">
    <?= \View::forge('noviusos_blognews::front/post/title', array('item' => $item)) ?>
    <?= \View::forge('noviusos_blognews::front/post/summary', array('item' => $item)) ?>
  </div>
  <div class="blognews_secondary_information">
    <?= \View::forge('noviusos_blognews::front/post/publication_date', array('item' => $item)) ?>
    <?= \View::forge('noviusos_blognews::front/post/tags', array('item' => $item)) ?>
  </div>
</div>
```

We deleted the thumbnail, author, categories and comment count from this view file.

2nd solution: extends the configuration

The blog application allows to disable some elements from its configuration. In our situation, it's possible for every elements we don't want to display, except the thumbnail.

When using this blog configuration file, it acts on both the list and the full item view, which is not really what we want (so this solution is just shown as an example).

Thanks to the cascading file system, we can copy the original `noviusos_blognews::config/config.php` file in our local directory: `local::config/apps/noviusos_blognews/config.php`

```
<?php

// We only keep the keys we want to alter
return array(
    'categories' => array(
        'show' => false,
    ),
    'authors' => array(
        'show' => false,
    ),
    'comments' => array(
        'show' => false,
    ),
);
```

Adding the CSS

1st solution: extending the view

We create the `local::views/apps/noviusos_blognews/front/post/list.view.php` file:

```
<?php

// We add our custom CSS file
\Nos\Nos::main_controller::addCss('static/css/blog_custom.css');

// We include the original file (which displays the post list)
include APPPATH.'applications/noviusos_blognews/views/front/post/list.view.php';
```

Our altered view first include a CSS file (to be created in `public/static/css/blog_custom.css`), then calls the original view.

2nd solution: directly take action on the template

It's also possible to include the CSS file with the `front.start` event, but in this case, it will be included on every pages of your website, not only on the blog page.

In the `local/bootstrap.php` file (create it if necessary):

```
<?php

// This event is triggered when loading a page of the website
Event::register('front.start', function() {
    \Nos\Nos::main_controller::addCss('css/blog_custom.css');
});
```

For the [Novius OS](#) website, we created our own templates, which are bundled with the appropriate CSS files to change how the blog is displayed.

1.4.5 Add an action in the admin

Actions are defined in the `config/common/{model}.config.php` file.

The best way to proceed is to be inspired by the default existing actions of Novius OS.

Placeholders

Action's configuration contains `{{placeholders}}`.

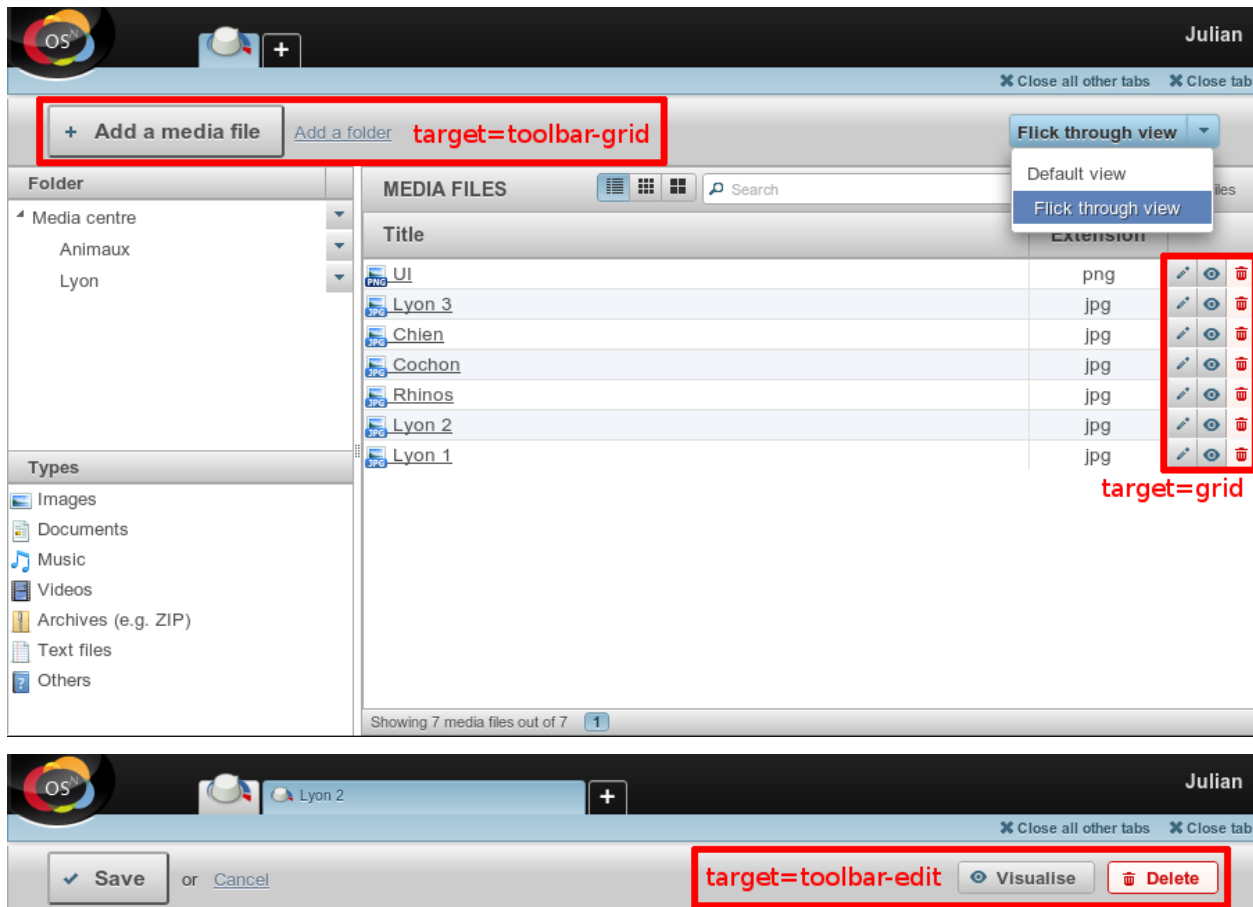
- Replaced with **PHP** :
 - `{{model_label}}`: the model name
 - `{{controller_base_url}}`: URL of the model's controller
- Replaced by the **App Desk (JavaScript)** :
 - `{{context}}`: current context (or first one when several are shown)


Every other placeholders are replaced according to the data of the **item**: `{{_id}}` and `{{_title}}` in this case, but also any field defined in the `data_mapping`.

Action's target

There are 3 possibles targets for the actions:

- **toolbar-grid**: App Desk's toolbar
- **grid**: item line in the main grid of the App Desk
- **toolbar-edit**: toolbar on the editing / addition form





Lyon 2

Change the file: Aucun fichier choisi

SEO, Media URL: .jpg ☐ Use title

Select a folder where to put your media file:

- Media centre
 - Animaux
 - Lyon**

Add / Edit / Delete

add action:

- opens a new tab ;
- calls the `action_insert_update()` method on the `Nos\Media\Controller_Admin_Media` controller ;

- with the `$_GET['context']` parameter, allowing to pre-select the active context ;
- is only shown in the App Desk's toolbar.

edit action:

- opens the edition form (the method is not specified for `nosTabs`, so the default open value will be used: it will focus the existing tab if it's already opened, or will create a new one otherwise ;
- calls the `action_insert_update($id)` method on the `Nos\Media\Controller_Admin_Media` controller ;
- with an `id` parameter ;
- is only shown in the main grid.

delete action:

- calls the `action_delete($id)` method on the `Nos\Media\Controller_Admin_Media` controller ;
- with an `id` parameter ;
- is shown both in the main grid and the edition form, but only for existing items (not for adding new items).

```
<?php
```

```
return array(  
  
    // Default ADD action  
    'add' => array(  
        'label' => __('Add {{model_label}}'),  
        'primary' => true,  
        // Opens a new tab on click  
        'action' => array(  
            'action' => 'nosTabs',  
            'method' => 'add',  
            'tab' => array(  
                'url' => '{{controller_base_url}}insert_update?context={{context}}',  
            ),  
        ),  
        // The ation is only be shown in the App Desk's toolbar  
        'targets' => array(  
            'toolbar-grid' => true,  
        ),  
    ),  
  
    // default EDIT action  
    'edit' => array(  
        'label' => __('Edit'),  
        'primary' => true,  
        'icon' => 'pencil',  
        // Opens the item on click (will refocus the tab when existing)  
        'action' => array(  
            'action' => 'nosTabs',  
            'tab' => array(  
                'url' => "{{controller_base_url}}insert_update/{{_id}}",  
                'label' => '{{_title}}',  
            ),  
        ),  
        // The action is only be shown in the main grid  
        'targets' => array(  

```

```

        'grid' => true,
    ),
),

// Default DELETE action
'delete' => array(
    'label' => __('Delete'),
    'primary' => true,
    'icon' => 'trash',
    'red' => true,
    // Opens a confirmation popup on click
    'action' => array(
        'action' => 'confirmationDialog',
        'dialog' => array(
            'contentUrl' => '{{controller_base_url}}delete/{{_id}}',
            'title' => strtr($config['i18n']['deleting item title'], array(
                '{{title}}' => '{{_title}}',
            )),
        ),
    ),
),
// The action is shown both in the main grid and the edition form...
'targets' => array(
    'grid' => true,
    'toolbar-edit' => true,
),
// ...but not for new items!
'visible' => function($params) {
    return !isset($params['item']) || !$params['item']->is_new();
},
),
);

```

1.4.6 Alter a behaviour of the front-office

Novius OS provides an event-base mechanism to interact with the core.

There are 2 types of events:

- Those which can alter data ;
- Those which only notify an action occurred.

<?php

```

// Exemple of notification event
Event::register('event_name', function($value)
{
    // L'action 'event_name' s'est produite
});

```

<?php

```

// Exemple of an event which may alter data
Event::register_function('event_name', function(&$value)
{
    // $value peut être modifiée
});

```

All events are documented in the API.

See Also:

Events

Redirecting depending on the URL

It's possible to create an *External link* page from the back-office to make a 301-redirect.

It's also possible to configure 301-redirects using the `./htaccess` file.

Last, it's also possible to do it from the source code, like below:

```
<?php

Event::register_function('front.start', function($params)
{
    // The Str class is from FuelPHP
    if (Str::starts_with($params['url'], 'an-old-url'))
    {
        // Note: 10 == strlen('an-old-url')
        $new_url = 'my-new-url'.substr($params['url'], 10);

        // The Response class is from FuelPHP
        Response::redirect($new_url, 'location', 301);
    }
});
```

Sending a thank-you mail from a contact form

```
<?php

Event::register_function('noviusos_form::after_submission', function(&$answer, $enhancer_args)
{
    foreach ($answer->fields as $field)
    {
        if ($field->anfi_field_type == 'email' && !empty($field->anfi_value)
        {
            $email = Email::forge();
            $email->from('my@email.me', 'My email');
            $email->to($field->anfi_value);
            $email->subject('Your contact request');

            // Textual email (use html_body() instead if you want to send HTML email)
            $email->body('Thank you for contacting us. We received it and will answer to you soon.');
```

```
            try
            {
                $email->send();
            }
            catch(\Exception $e)
            {
                // Could not send the email
            }
        }
    }
});
```


1.5 Version notes

1.5.1 Release notes 0.2

New features

- Novius OS can now manage multiple contexts: one or several websites, each one with one or several languages
- Added the **Slideshow** application
- Added the **Form** application
- Added the **‘Build your app’ wizard** application
- Added the **Simple Google+ share** `sharer` on the same level as Facebook and Twitter
- Back-office can now speak french in addition to english
- Back-office tabs can be closed with a middle click
New button *Close all other tabs*

Developpers

- Consequences of the switch from multi-languages to multi-contexts
 - The context configuration is done in a dedicated file (it’s not in `config.php` anymore). Two new keys `contexts` and `sites` exists now, in addition to `locales`
 - Every columns `lang`, `lang_common_id`, `lang_is_main` of the database have been renamed with `context`
 - The new `context` columns are now larger, from 5 to 25 characters
 - The `Translatable` behaviour has been renamed `Twinnable`
 - In the CRUD, the `context` notion has been replaced by `environment`, to prevent confusions (`context_relation` -> `environment_relation`, `item_context` -> `item_environment`)
 - Every associated variables have been renamed, too
- Updated vendors to the following versions:
 - **jQuery**, updated from 1.7.2 to **1.8.2**
 - **jQuery UI**, updated from 1.8.22 to **1.8.24**
 - **Wijmo**, updated from 2.1.4 to **2.2.2**
 - **tinyMCE**, updated from 3.5.6 to **3.5.7**
 - **FuelPHP** and its packages (including `email`), updated from 1.2 to **1.4**
- Changes in the pages API:
 - New class `Tools_Url`
 - `Model_Page->get_link()` -> `Model_Page->link()`
 - `Model_Page->get_href()` -> `Model_Page->url()`
 - `Model_Page::get_url()` -> `Tools_Url::page()`
 - Deleted `Model_Page::get_url_absolute()`

- Every methods now returns absolutes URLs
- Merged and enhanced configuration files for app-desk, inspector and CRUD:
 - New common configuration file for model-specific data
 - Possibility to format a grid column from the PHP configuration (not only JavaScript).
 - Possibilité de formater une colonne d'une grid via la configuration PHP (et plus seulement en Javascript)
- In `Controller_Crud`, the `from_item` was renamed to `init_item` and is only called for new items
- New `Attachment` class to handle files attached to an item, without adding them to the media centre
- `widget` have been replaced to `renderers`. Classes and view files were updated accordingly.
- Every view and configuration file are extendable in the `local/config` directory from the website
- Created a new controller for enhancer's popup in the Wysiwyg, with a default preview
- The `upload.disabled_extensions` configuration key was moved to `novius-os.upload.disabled_extensions`
- `$page` and `$main_controller` variables are now available within the template
- The time picker `renderer` can be used outside a `Fieldset` (standalone way)
- The `front.start` event has an additional `cache_path` parameter.

1.5.2 Migration guide from the 0.1 version to the 0.2 version

Mandatory breaking changes

Consequences of the switch from multi-languages to multi-contexts

- The context configuration is done in a dedicated file (it's not in `config.php` anymore). Two new keys `contexts` and `sites` exists now, in addition to `locales`
- Every columns `lang`, `lang_common_id`, `lang_is_main` of the database have been renamed with `context`
- The new context columns are now larger, from 5 to 25 characters
- The `Translatable` behaviour has been renamed `Twinnable`
- In the CRUD, the context notion has been replaced by `environment`, to prevent confusions (`context_relation` -> `environment_relation`, `item_context` -> `item_environment`)
- Every associated variables have been renamed, too

Changes in the pages API

- `Model_Page->get_link()` -> `Model_Page->link()`
- `Model_Page->get_href()` -> `Model_Page->url()`
- `Model_Page::get_url()` -> `Tools_Url::page()`
- Deleted `Model_Page::get_url_absolute()`
- Every methods now returns absolutes URLs

Changes in the events API

- `front.start` now takes an array as argument, containing the `url` key

Procedure to follow

- Create the `contexts.config.php` (based on the sample).
- Change `config.php` and add the `novius-os` key (based on the sample).
- Search `_lang`, replace it with `_context` (i.e. `page_lang => page_context`)
- Search `Nos\Model_`, replace it with `Nos\{{app}}\Model_` (i.e. `Nos\Model_Page => Nos\Page\Model_Page`, `Nos\Model_Media => Nos\Media\Model_Media`)
- Search `Translatable`, replace it with `Contextable`
- Search `Model_Page->get_link()`, replace it with `Model_Page->link()`
- Search `Model_Page->get_href()`, replace it with `Model_Page->url()`
- Search `Model_Page::get_url()`, replace it with `Tools_Url::page()`
- Search `Nos\Widget_Page_Selector`, replace it with `Nos\Page\Renderer_Selector`
- Search `Nos\Widget_Media`, replace it with `Nos\Renderer_Media`
- Search `Nos\Widget_`, replace it with `Nos\{{Renderer_Version}}`

Changes in the applications

Metadata

- Launchers: the `url` key has been replaced by `action` (standard *nosAction* syntax)
- Launchers: the `icon64` key has been replaced by `icon`

```
<?php

return array(
    'launcher_name' => array(
        'url' => '{{your_url_here}}',
    ),
);

// Replace with

return array(
    'launcher_name' => array(
        'action' => array(
            'action' => 'nosTabs',
            'tab' => array(
                'url' => '{{your_url_here}}',
            ),
        ),
    ),
);
```

- An application now proved a list of icons:

```
<?php

return array(
    'icons' => array(
        64 => 'static/apps/noviusos_page/img/64/page.png',
        32 => 'static/apps/noviusos_page/img/32/page.png',
        16 => 'static/apps/noviusos_page/img/16/page.png',
    ),
);
```

- A launcher without `icon` will use the 64-icon from its application
- A tab without `iconUrl` will use the 32-icon from its application
- An enhancer without `iconUrl` will use the 16-icon from its application

CRUD configuration

- `widget` was renamed to `renderer`

```
<?php

return array(
    'field_name' => array(
        'widget' => 'Nos\Widget_Media',
        'widget_options' => array(),
    ),
);

// À remplacer par :
return array(
    'field_name' => array(
        'renderer' => 'Nos\Renderer_Media',
        'renderer_options' => array(),
    ),
);
```

“Full” 0.2 migration

This section relies on the hypothetical existence of a `lib_agenda` application.

Appdesk

Models have a new `common` configuration file which contains: * a `data_mapping` * a list of actions

In `appdesk.config.php`:

- Delete the `selectedView` and `views` keys (if you only have one view with no JS config file).
- Spot the main model of your appdesk (it's in the `query.model` key).
- **Create the associated `config/common/{model_name}.config.php` file**
 - `{{model_name}}` relates to the name of the model, in lowercase and without the `Model_` prefix (for instance, `Model_Page` becomes `page`)
 - `Model_Page` will relates to the `config/common/page.config.php` file

Note: Pay attention to have 'hideContexts' => true, in your appdesk's configuration if you items are not Contextable.

Data mapping data_mapping is the merge of the dataset and appdesk.grid.columns keys.

```
<?php

// Old code of appdesk.config.php
return array(
    'query' => array(
        'model' => 'Lib\Agenda\Model_Event',
        'order_by' => array('evt_date_begin' => 'DESC'),
        'limit' => 20,
    ),
    // ...
    'dataset' => array(
        'id' => 'evt_id',
        'title' => 'evt_title',
        'periode' => array(
            'search_column' => 'evt_date_begin',
            'dataType' => 'datetime',
            'value' => function ($object) {
                // ...
            },
        ),
    ),
    // ...
    'appdesk' => array(
        // ...
        'grid' => array(
            'urlJson' => 'admin/lib_agenda/appdesk/json',
            'columns' => array(
                'id' => array(
                    'headerText' => __('Id'),
                    'dataKey' => 'id'
                ),
                'title' => array(
                    'headerText' => __('Name'),
                    'dataKey' => 'title'
                ),
                'periode' => array(
                    'headerText' => __('Dates'),
                    'dataKey' => 'periode'
                ),
                'published' => array(
                    'headerText' => __('Status'),
                    'dataKey' => 'publication_status'
                ),
                'actions' => array(
                    'actions' => array('update', 'delete'),
                ),
            ),
        ),
        // ...
    ),
);
```

```
<?php

// New code in appdesk.config.php
return array(
    'query' => array(
        'order_by' => array('evt_date_begin' => 'DESC'),
        'limit' => 20,
    ),
    // Tells what is the model, and wich 'common' file to load
    'model' => 'Lib\Agenda\Model_Event',
    // ...
    // MOVE / MERGE the 'dataset' key into
    // ...
    'appdesk' => array(
        // ...
        // MOVE / MERGE the 'grid' key into
        // ...
    ),
);
```

```
<?php

// Code in the new 'event.config.php' file
return array(
    // Merge of 'appdesk.dataset' and 'appdesk.grid.columns'
    'data_mapping' => array(
        'id' => array(
            'title' => __('Id'),
            'column' => 'evt_id'
        ),
        'title' => array(
            'title' => __('Name'),
            'column' => 'evt_title'
        ),
        'periode' => array(
            'title' => __('Dates'),
            'search_column' => 'evt_date_begin',
            'value' => function ($object) {
                // ...
            }
        ),
        'published' => array(
            'title' => __('Status'),
            'method' => 'publication_status'
        ),
    ),
);
```

Some remarks: * headerText can be written title (easier to remember, used in the natives apps) * datakey can be written "column" * value can still contain a callback function * method is a new option which execute a method instead of retrieving a column

Actions Actions of the main model (the one on the grid) must also be moved in the common file.

```
<?php
```

```
// Old code of appdesk.config.php
return array(
    // ...
    'appdesk' => array(
        // ...
        // MOVE the 'actions' key into 'config/common/{{model_name}}.config.php'
        'actions' => array(
            'edit' => array(
                // ...
            ),
            'delete' => array(
                // ...
            ),
        ),
        // ...
    ),
    // ...
);

<?php

// new code in appdesk.config.php
return array(
    // ...
    'appdesk' => array(
        // ...
        // The 'actions' key is not here anymore
        // ...
    ),
    // ...
);

<?php

// New code in 'config/common/event.config.php'
return array(
    'data_mapping' => array(
        // ...
    ),

    // Configuration array moved from 'appdesk.actions'
    'actions' => array(
        'Lib\Agenda\Model_Event.edit' => array(
            // ...
        ),
        'Lib\Agenda\Model_Event.delete' => array(
            // ...
        ),
    ),
);
```

From the moment the common file is used, the following generic actions appear: * add * edit (and not update !) * visualise (if appropriate, i.e. if the model has the `Urlrenhancer` behaviour) * delete * share (if appropriate)

In our `lib_agenda` application, `Model_Event` had an update action, which now appears twice... (because we used the wrong name update instead of edit).

So, to fix it in our `lib_agenda` application, we need to: * rename update to edit * Delete the edit and delete

keys, since we're doing the default processing * It's possible to keep only the keys we need to overwrite (to change the default texts for instance...)

Notes : * In the Novius OS version used, it was needed to prefix actions by the model name, it's no longer necessary in the final version * `{{controller_base_url}}` can be used in action's URLs. In the `lib_agenda` application, it's replaced by `lib_agenda/admin/agenda/` * A new `targets` keys allows to define where actions must appear (see comments).

```
<?php

// Example of placeholder {{controller_base_url}} + 'targets'
array(
    'Lib\Agenda\Model_Event.edit' => array(
        'action' =>
            array(
                'action' => 'nosTabs',
                'tab' => array(
                    'url' => "{{controller_base_url}}insert_update/{{id}}",
                    'label' => __('Modifier'),
                ),
            ),
        'label' => __('Modifier'),
        'primary' => true,
        'icon' => 'pencil',
        // New key to define where this action appear
        'targets' => array(
            'grid' => true, // In the grid (in the last 'actions' column)
            'toolbar-grid' => true, // On the appdesk, in the toolbar (previously configured using '
            'toolbar-edit' => true, // On the item form (top-right corner)
        ),
    ),
);
```

The default configuration for the targets is like below: * `grid`: edit + visualise + delete * `toolbar-grid`: add * `toolbar-edit`: visualise + share + delete

Note: For now, `appdesk.appdesk.buttons` is still defined, and has priority over the default configuration. Given we have 2 actions 'Add an event' and 'Add a category' from 2 different models, we can't delete it (for now).

I18n and translations Texts can be configured using the `i18n` key.

Please see the `framework/config/common_i18n.config.php` to get the list of possible keys.

```
<?php

return array(
    'i18n' => array(
        // Crud
        'notification item added' => __('And voilà! The page has been added.'),
        'notification item deleted' => __('The page has been deleted.'),

        // General errors
        'notification item does not exist anymore' => __('This page doesn't exist any more. It has be
        'notification item not found' => __('We cannot find this page.'),

        // Blank slate
        'translate error parent not available in context' => __('We're afraid this page cannot be ad
        'translate error parent not available in language' => __('We're afraid this page cannot be a
```



```

// Deletion popup
'deleting item title' => __('Deleting the page '{{title}}'),

# Delete action's labels
'deleting button 1 item' => __('Yes, delete this page'),
'deleting button N items' => __('Yes, delete these {{count}} pages'),

'1 item' => __('1 page'),
'N items' => __('{{count}} pages'),

# Keep only if the model has the behaviour Contextable
'deleting with N contexts' => __('This page exists in <strong>{{context_count}} contexts</strong>'),
'deleting with N languages' => __('This page exists in <strong>{{language_count}} languages</strong>'),

# Keep only if the model has the behaviours Contextable + Tree
'deleting with N contexts and N children' => __('This page exists in <strong>{{context_count}} contexts and {{children_count}} children</strong>'),
'deleting with N contexts and 1 child' => __('This page exists in <strong>{{context_count}} contexts and 1 child</strong>'),
'deleting with N languages and N children' => __('This page exists in <strong>{{language_count}} languages and {{children_count}} children</strong>'),
'deleting with N languages and 1 child' => __('This page exists in <strong>{{language_count}} languages and 1 child</strong>'),

# Keep only if the model has the behaviour Tree
'deleting with 1 child' => __('This page has <strong>1 sub-page</strong>.'),
'deleting with N children' => __('This page has <strong>{{children_count}} sub-pages</strong>.'),
),
);

```

Inspectors In the 0.1 version, the inspectors are configured in 3 different places: * The appdesk.appdesk.inspectors key * The inputs key * The inspector/{model}.config.php configuration file

In the 0.2 version, inputs must be moved into their corresponding inspector/{model}.config.php file.

Category

```

<?php

// Old code from appdesk.config.php
return array(
    // ...
    'inputs' => array(
        // This input communicates with the filter for the category inspector
        // We move the key (evt_cat_id) into 'input.key' and the callback function into 'input.query'
        'evt_cat_id' => function($value, $query) {
            // ...
        },
    ),
    // ...
);

<?php

// New code in config/controller/admin/inspector/category.config.php
return array(
    // ...
    'input' => array(
        'key' => 'evt_cat_id',
        'query' => function($value, $query) {

```

```
        // ...
    },
),
// ...
);
```

Date

```
<?php
```

```
// Old code from appdesk.config.php
```

```
return array(
    // ...
    'appdesk' => array(
        'appdesk' => array(
            // ...
            'inspectors' => array(
                // This array must be moved in the configuration file of the inspector, under a new key
                'startdate' => array(
                    'label' => __('Start date'),
                    'url' => 'admin/lib_agenda/inspector/date/list',
                    'inputName' => 'startdate',
                    'vertical' => true
                ),
                // ...
            ),
        ),
    ),
    // ...
),
);
```

Here, the inspector doesn't have a configuration file yet, we need to create one:

```
<?php
```

```
// New file config/controller/admin/inspector/date.config.php
```

```
return array(
    'input' => array(
        'key' => 'evt_date_begin',
        // the 'geury' keys is not needed, the date inspector will generate it automatically using the
    ),

    // Old 'appdesk.appdesk.inspectors.startdate'
    'appdesk' => array(
        'label' => __('Start date'),
    ),
);
```

The idea is to encapsulate the `appdesk.appdesk.inspectors.{{inspector_name}}` array inside an `appdesk` key of the inspector's configuration file.

published

```
<?php
```

```
// Old code from appdesk.config.php
```

```
return array(
    // ...
```

```

    'inputs' => array(
        // ...
        'evt_published' => function($value, $query) {
            // ...
        },
    ),
    // ...
    'appdesk' => array(
        'appdesk' => array(
            // ...
            'inspectors' => array(
                'published' => array(
                    'vertical' => true,
                    'url' => 'admin/lib_agenda/inspector/published/list',
                    'inputName' => 'evt_published',
                    'grid' => array(
                        'columns' => array(
                            'title' => array(
                                'visible' => false,
                                'dataKey' => 'title',
                            ),
                            'icon_title' => array(
                                'headerText' => __('Status'),
                                'dataKey' => 'icon_title',
                            ),
                        ),
                        'id' => array(
                            'visible' => false,
                            'dataKey' => 'id',
                        ),
                    ),
                ),
            ),
        ),
    ),
    // ...
),
// ...
);

```

The published inspector already has its configuration file. Let's complete it with a new appdesk key:

```

<?php

// New file config/controller/admin/inspector/date.config.php
return array(
    'data' => array(
        // ...
    ),

    // Old 'appdesk.appdesk.inspectors.published'
    'input' => array(
        'key' => 'evt_published',
        'query' => function($value, $query) {
            // ...
        },
    ),

    // Old 'input.evt_published'

```

```
'appdesk' => array(
    'vertical' => true,
    'inputName' => 'evt_published',
    'url' => 'admin/lib_agenda/inspector/published/list',
    'grid' => array(
        'columns' => array(
            'title' => array(
                'visible' => false,
                'dataKey' => 'title',
            ),
            'icon_title' => array(
                'headerText' => __('Status'),
                'dataKey' => 'icon_title',
            ),
            'id' => array(
                'visible' => false,
                'dataKey' => 'id',
            ),
        ),
    ),
);
```

1.5.3 Release notes Chiba 1

New features

- The behaviour publishable (Behaviour_Publishable) now allows to choose publication start and end dates.

Form

- Adding a progress bar on multi-page form.

Blog / News

- Display related posts of authors.

Developer

- Front improvements
 - Profiling is activated by default on front in the DEVELOPMENT environment.
 - Setting configuration `novius-os.cache` by default always at true.
 - Setting configurations `novius-os.cache_duration_page` and `novius-os.cache_duration_function` at 600 by default, except in PRODUCTION at 3600.
 - New events: `front.pageFound` and `front.response`.
 - New methods in `Controller_Front`: `getContext`, `disableCaching`, `setCacheDuration`, `setStatus`, `setHeader`, `getCustomData`, `setCustomData`, `sendContent`, `addCacheSuffixHandler`.
 - Status and headers are now save in cache.

- Mechanism to adapt the cache path with suffixes, depending GET parameters or what you want (with callables).
- Mechanism to execute code when using the cache.
- Models properties
 - All models (core and native apps) now defines the `$_properties`
 - Implement a cache mechanism on models properties, using FuelPHP cache. Attempt to auto-refresh when an unknown properties is called.
- Migrations are now dispatched per application.
- New `metadata key requires` which allows to define that an application requires another one.
- It is now possible to use `href="##..."` in enhancers or templates; occurrences will be replaced by `href="#..."` without prepending the `base_url`.
- CRUD: When returning a string in the `disabled` key, it is displayed as a title. `disabled` and `visible` keys can now be simple values, callbacks or array of callbacks.
- Resized images are now secured: you can't generate a lot of thumbnails and flood the server anymore.
- Permissions
 - Ability to define per-application permissions with a configuration file
 - New API to check permissions for a user, or a specific role
 - Ability to enable multi-roles on the users with the `novius-os.users.enable_roles` configuration.

Deprecated

- Moved `Nos\Renderer_Media` to `Nos\Media\Renderer_Media`.
- Launchers configuration: the `url` key is deprecated. Use `action` instead.
- The `widget` key is deprecated in renderer configuration. Use the `renderer` key and update the class name.
- The `widget_options` key is deprecated in renderer configuration. Use the `renderer_options` key.
- `\Config::extendable_load()` is deprecated. Renamed to `\Config::loadConfiguration()`.
- `Orm_Behaviour_Publishable` configuration: the `publication_bool_property` key is deprecated. Use `publication_state_property` instead.

1.5.4 Migration guide from the 0.2 version to the Chiba 1 version

Few changes are needed to migrate to the next version. The new API is compatible with the old one. From this version, Novius OS handles depreciations. Here a deprecated items from Chiba 1:

- `Nos\Renderer_Media` should be renamed to `Nos\Media\Renderer_Media`.
- Launchers configuration: the `url` key is deprecated. Use 'action' instead.
- The `widget` key is deprecated in renderer configuration. Use the `renderer` key and update the class name.
- The `widget_options` key is deprecated in renderer configuration. Use the `renderer_options` key.
- `\Config::extendable_load()` is deprecated. Renamed to `\Config::loadConfiguration()`.
- `Orm_Behaviour_Publishable` configuration: the `publication_bool_property` key is deprecated. Use `publication_state_property` instead.

Pages are now cached by default (for 10 minutes) ; it could result to unexpected behaviour for custom applications using enhancer and / or urlenhancer.

Some migration files need to be executed. If you use `oil`, the new command is `php oil refine migrate -m` as migration files are now organised differently.

1.6 Contribute to Novius OS

1.6.1 Translate Novius OS

Novius OS is ready to be translated into any language (It has already been translated into French). Here is the process to translate Novius OS into your language.

Remember [we are here to help](#). Your contribution is very appreciated, thank you.

Copy style guide

We take copywriting and translation very seriously. We have therefore established guidelines common to all applications and languages in order to provide every Novius OS user with consistent and enjoyable copy.

- *Copy style guide (English)*
- *Charte rédactionnelle (Français)*
-

The copy style guide is to be read by every translator. If it's not available in your language then it is the first document to translate.

Note: We're happy to have the style guide translated by a professional translator. This key document may indeed prove hard to localise. [Contact us](#) about this translation.

Translation server

Novius OS translation server is available at translate.novius-os.org. It is powered by [Pootle](#). Here is a quick guide to Pootle. (You may also want to have a look at [Evernote's handy tour of Pootle](#).)

Available languages

Languages

| Language ▾ | Progress ▾ | Last Activity ▾ |
|------------|------------------------|---|
| French | <div><div></div></div> | 2013-02-19 14:42 (lefeuvre) |

If on translate.novius-os.org home page, your language is not listed under *Languages*, just [contact us](#). We'll add it for you.

Translation files

Once your language is available on translate.novius-os.org, click it. Then click the version of Novius OS you want to translate. If you don't know which one to choose, [ask us](#), we'll tell you.

You now see a list of directories. The directories starting with `noviusos_` are applications. `framework` contains the strings from the core.

Files and Subfolders

| Name ↕ | Progress ↕ | Total ↕ | Need Translation ↕ | Suggestions ↕ |
|---------------------------------------|------------------------|---------|--------------------|---------------|
| 📁 framework | <div><div></div></div> | 971 | 0 | |
| 📁 noviusos_appmanager | <div><div></div></div> | 60 | 0 | |
| 📁 noviusos_appwizard | <div><div></div></div> | 244 | 208 | |
| 📁 noviusos_blog | <div><div></div></div> | 81 | 0 | |
| 📁 noviusos_blognews | <div><div></div></div> | 318 | 0 | |
| 📁 noviusos_comments | <div><div></div></div> | 12 | 0 | |

To start translating, don't click *Continue translation* but a number in the *Need translation* column. This allows you to choose a directory. Please follow these priorities:

- Top priority, the core: `framework`.
- Then the native applications: Webpages (`noviusos_pages`), Media Centre (`noviusos_media`), Users (`noviusos_user`) and Applications manager (`noviusos_appmanager`).
- And finally the non-native applications.

Suggesting and submitting translations

Everybody can suggest translations. A suggestion will be reviewed by an approved translator before being submitted. Only submitted translations are applied to Novius OS. Unreviewed or rejected suggestions stay in Pootle.

Note: You don't even need to create an account to suggest translations. Nevertheless creating an account only takes you a minute and allows you to select your languages and avoid the CAPTCHA protection.

Being an approved translators is a great way to make a difference to the Novius OS project. Everyone can apply: Just [drop us](#) a few lines about yourself and include your Pootle username. We'll review your application before granting you extended permissions.

When translating

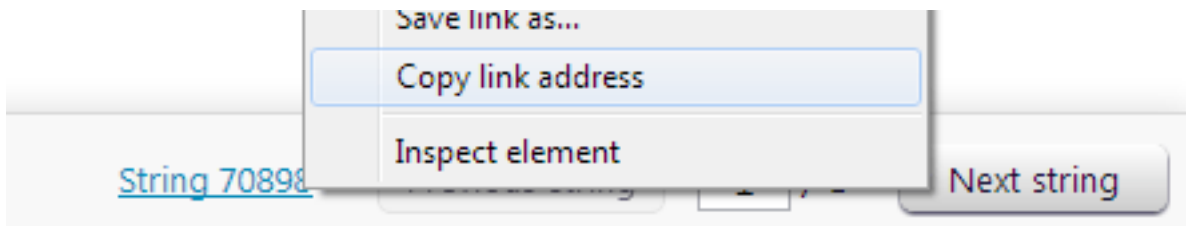
Placeholders and tags

- Some strings include variables, e.g. `'Welcome back, {{user}}'`. Obviously, Novius OS end users don't see these variables which are replaced by the actual value, e.g. `'Welcome back, Joe'`. Variables are to be kept, untranslated, e.g. `'Re-bonjour {{user}}'`.
- HTML tags are also to be kept and not to be translated. In most cases, you'll find a start tag and an end tag, e.g. `'This page has one sub-page'`. Text between tags must be translated, e.g. `'Cette page a une sous-page'`.
- For non-breaking spaces, please use the HTML entity, i.e. ` `.

Dispelling doubts

Don't translate in the dark! When you don't understand a string, need to know the context or have any doubt, please [contact us](#). Our job is to make translation easy. We're happy to add notes to translator in the translation files for others to benefit from your feedback. We can also provide you with screenshots or indications to find a string in the UI of Novius OS.

When your question or comment regards a specific string, please give us the link to the string. You'll find it the bottom right corner:



Translating the documentation

This documentation is powered by [Read The Docs](#) which uses Rich Structured Text (RST) files. This format is human-readable and therefore easier to translate.

If on this [page](#), your language is not listed under *Translations*, [contact us](#). We'll set up the GitHub repo for you.

Once the repo is ready, clone it. You may also want to clone the [English repo](#) so you can copy files from the original version to the translation.

That's it, you're all set to start translating. Thank you very much for your contribution!

1.6.2 Copy style guide (English)

Introduction

This style guide is intended for all persons writing copy for Novius OS: applications developers and designers, contributors to the core, translators. It establishes guidelines common to all applications and languages in order to **provide every Novius OS user with consistent and enjoyable copy**.

This guide is based on Aaron Walter's [design personas](#). MailChimp's [Voice and Tone](#) guide, created by Walter's team, has also been a source of inspiration.

About translation

Translation is more than just a word-to-word job. One must remain faithful to the original design while adapting it for a new audience, for a different culture. A literal translation of Novius OS copy does not make sense. The translation must read as if the copy had originally been written in the target language. The first document to translate are these guidelines so that a **localised style guide** is available.

Personality and tone

Novius OS is **designed for professionals**. They don't use it for fun, they have a job to do. Novius OS must show its users they share a common goal: to get the job done as efficiently as possible. Novius OS therefore adopts a professional and straight-to-the-point tone.

This said, **using Novius OS doesn't have to be boring**. The usual flavourless software speech (e.g. Please enter a valid value) is to be avoided. We don't Novius OS to be seen as yet another tool.

Novius OS can make its users smile—especially when they succeeded in carrying out their job—but not laugh. The software is not their friend. It is rather **a trustworthy and knowledgeable colleague they're happy to work with**. A team of colleagues actually, as Novius OS says 'We' not 'I'.

Finally, although many Novius OS followers are developers, copy must be jargon-free. What's more, bro-ish tone (e.g. Sorry dude, I screwed up!) is inappropriate.

Personality traits

- **Professionnal** but not boring.
- **Straight-to-the-point** but not bossy.
- **Consistent** but not repetitive.
- **Friendly** but not bro-ish.
- **Knowledgeable** but not haughty.

English language

Novius OS is written in **British English**. So could you please be kind enough to restrain from using the spelling 'Media Center' instead of the more gentrified 'Media Centre'?

For quotation marks, we follow [the rule given by Oxford Dictionnaries](#)—single inverted commas e.g. 'Hello world'. Besides please use apostrophes (') not the prime symbol (').

Copy examples

Actions:

Sign in Let's get started!

Log out Sign out (see you!)

Add new item Add a new page ¹

Save an item Save ¹

Success:

New item added All done! The blog post has been added. ²

Item updated OK, changes are saved. ²

Errors:

User mistake You must add a title for the product to be saved. Sorry about that! ³

System error Something went wrong. Please try again and contact your developer or Novius OS if the problem persists. We apologise for the inconvenience caused. ⁴

¹ No need to say more. We want clearly labelled primary actions.

² A little diversity is welcome as long as the style is consistent. E.g. 'There you go!', 'All wrapped up!' or 'Great!' to start a success message.

³ Don't assume this is the user's fault. Maybe it wasn't very clear the field was mandatory.

⁴ Don't 'Oops!' the user. She/he might have lost time or data.

Error prevention (warning) Answers to this form have already been received. Modifying the form may alter the collected data. ⁵

Error prevention (confirm button) Don't worry, I know what I'm doing. ⁵

⁵ When you need the user to pay attention, get her/him involved by turning the copy into a conversation.