Novius OS Documentation

Release 0.2

Novius

November 22, 2013

Contents

Novius OS' documentation is hosted and generated by Read The Docs. Sources are available in a repository on Git Hub. Do not hesitate to propose modifications (if you spot an error) via `Pull Request`. You're also very welcome if you want to take part in its translation. Contact us for further details.

CHAPTER 1

How-tos

1.1 How to install Novius OS

This guide provides instructions to run on Ubuntu. Be sure to adapt the commands for your own OS.

- *Step 1: download Novius OS source code*
  - *Method A: Git and GitHub*
  - *Method B: from a .zip file*
- *Step 2: Server configuration*
  - *Method A: you control the server*
  - *Method B: Shared hosting*

1.1.1 Step 1: download Novius OS source code

Method A: Git and GitHub

Step A-1: prerequisites - install GIT

```
sudo apt-get install git
```

Step A-2: clone the sample website repository

We use submodules, so be sure to fetch them properly. The `--recursive` option does everything you need.

```
cd ~
git clone --recursive git://github.com/novius-os/novius-os.git
sudo mv novius-os /var/www/
```

This will download a sample repository, with several submodules:

- novius-os : the core of Novius OS, which has other submodules, like fuel-core or fuel-orm ;
- Other submodules in the local/applications folder: blog, news and comments.

Step A-3 (optional): change the version you want to use (if you're gutsy)

We configured the cloning of the repository to point to the latest available release (it's **master/0.1.5*** at the time I'm writing this).

When we deploy a version, we create a new branch for it.
For now, we keep synchronised all the dependant repositories. Hence, an application provided on our Github will follow the same version number as the core. So if you're using novius-os/core version 0.3 (not yet released!), you need to use novius-os/app in the same version 0.3 too.

To change the version you're using after cloning, *don't forget to update the git submodules*.
Example to use the latest nightly from the 'dev' branch

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

Method B: from a .zip file

```
cd ~
wget http://nova.li/nos-015 -O novius-os.0.1.5.zip
unzip novius-os.0.1.5.zip
sudo mv novius-os /var/www/
```

Or download the file nos-015 and unzip it with your favourite program.

1.1.2 Step 2: Server configuration

We'll be discussing 2 cases:

- installation on a server you control (either your local machine, a VM or a dedicated external server) ;
- installation on a shared hosting service, without SSH command-line access or the possibility to change Apache configuration.

Method A: you control the server

Step A-1: make sure Apache's mod_rewrite is enabled

```
sudo a2enmod rewrite
```

Step A-2: configure a VirtualHost

Create a new `VirtualHost` for Novius OS (replace `nano` with your favourite text editor in the following commands)

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

Copy the following configuration in the file, and save it. Adapt the `ServerName` line with your own domain when installing on a production server.

```
<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 contains a *public* directory. The webroot should points to this directory.

Enable the freshly created `VirtualHost`

```
sudo a2ensite novius-os
```

Reload Apache (or your other web server) to take the new configuration into account.

```
sudo service apache2 reload
```

Step A-3: configure the `hosts` file, when installing on your local machine

If the `ServerName` is different than `localhost` (`novius-os` in the above example), you should add the server name into your `hosts` file.

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

Add the following line:

```
127.0.0.1   novius-os
```

Method B: Shared hosting

Step B-1: upload the source code to your server

You can choose the way you do it, depending on your shared hosting provider (FTP, SSH, Git...)

Step B-2: `.htaccess` files

Novius OS needs an `.htaccess` file to run.

In a classic installation, the `DOCUMENT_ROOT` should point to the `public` directory of Novius OS (see step A-2 above). On a shared hosting, you don't choose the location for `DOCUMENT_ROOT`. So you need to delete the `public/.htaccess` file and rename the `.htaccess.shared-hosting` inside the Novius OS root folder into `.htaccess`.

Then, edit this `.htaccess` file, and change the line beginning with `ErrorDocument` depending on where you installed Novius OS:

```
ErrorDocument 404 /novius-os-install-dir/public/htdocs/novius-os/404.php
```

If Novius OS has been installed in the root directory of your hosting:

```
ErrorDocument 404 /public/htdocs/novius-os/404.php
```

Step B-3: `local/config/config.php` file

Edit the `local/config/config.php` file, un-comment and adapt the following line to your case:

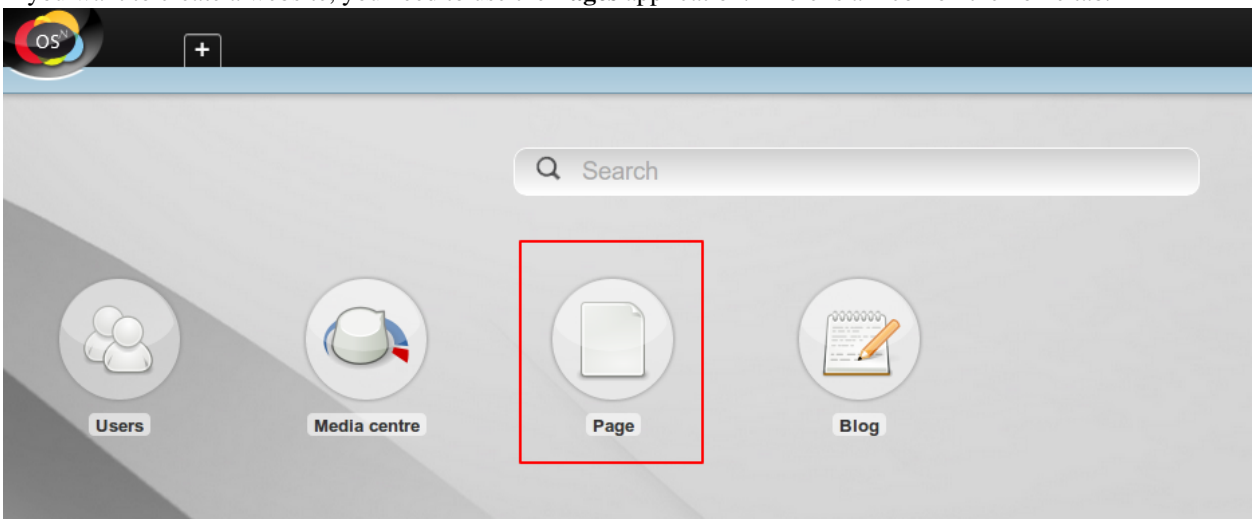
```
'base_url' => 'http://www.yourdomain.com/novius-os-install-dir/',
```

1.1.3 Step 3: Finish the installation

You've done the hardest. Now you just need to go through the `setup-wizard` to enjoy your Novius OS.

1.2 How to set up the front-office

If you want to create a website, you need to use the **Pages** application. There is an icon on the home tab.

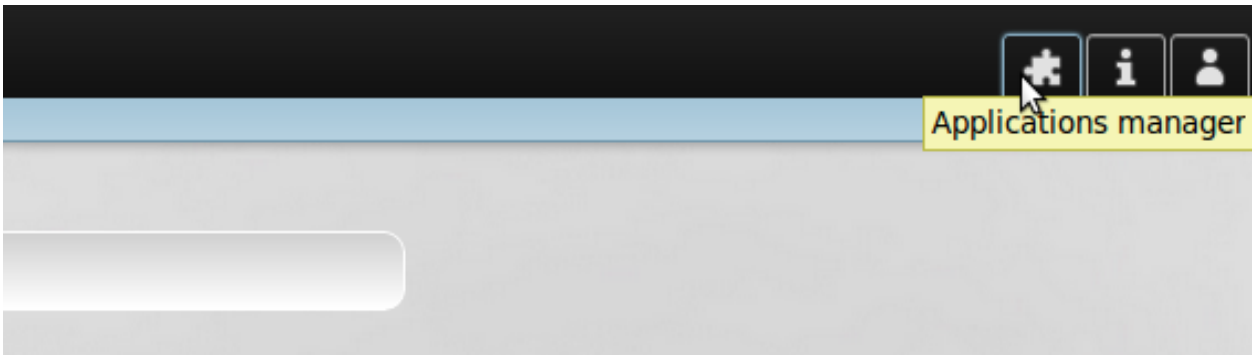


1.2.1 Templates

But pages need templates to define how they are displayed. So the first thing you need to do is installing a template. They are created by designers and developers.

If you just installed Novius OS, you can install a default template from the application manager:

1.2.2 Access the application manager

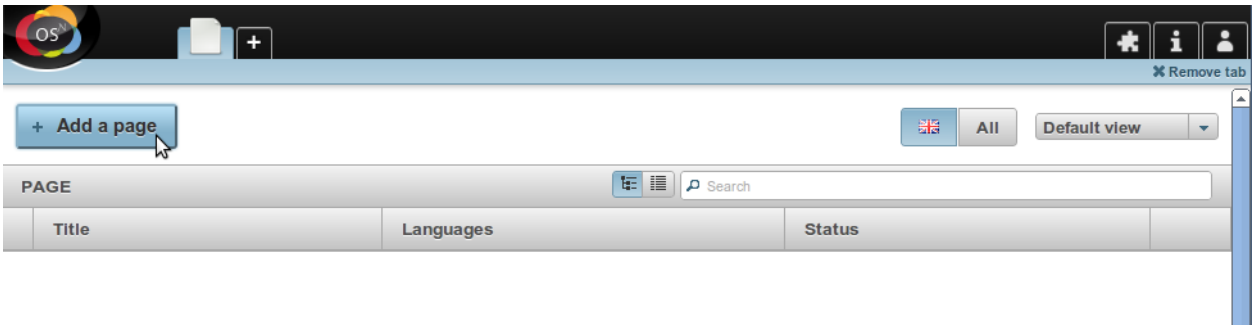


1.2.3 Install the application which provide the default template

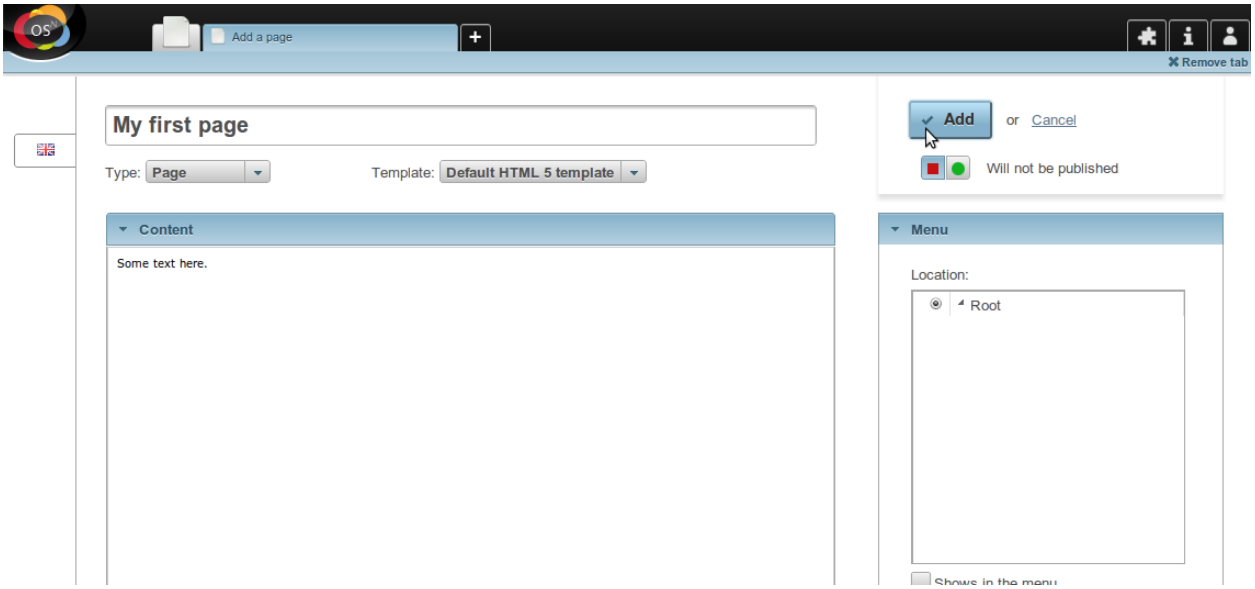
1.2.4 Open the page application

You can access it from the home tab (see above).

1.2.5 Add a page (your first one)

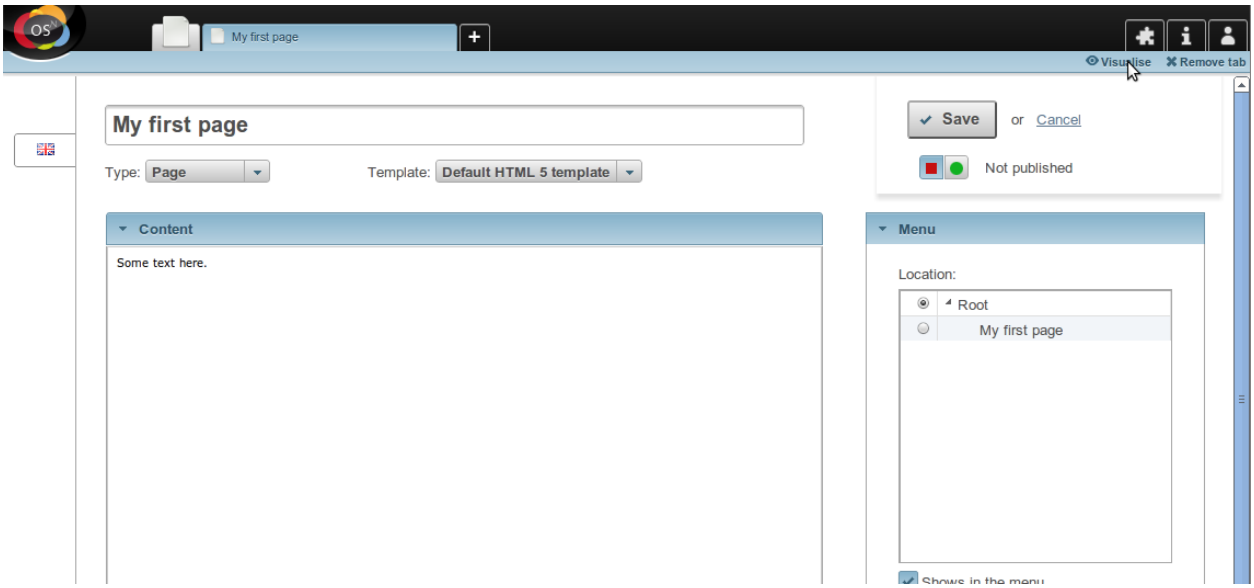


1.2.6 Write some content and “Add”



1.2.7 Preview what you’ve done

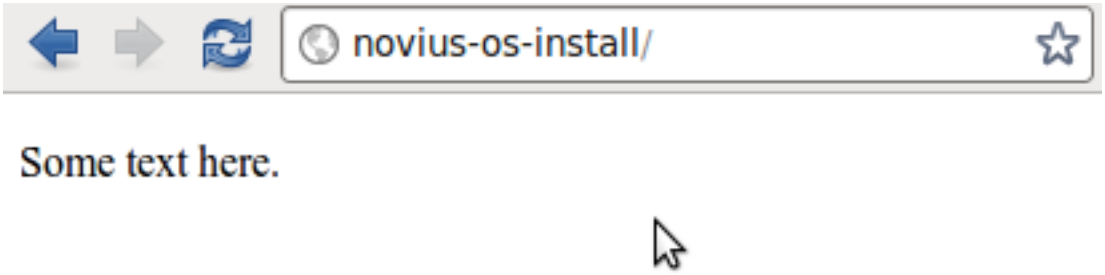
The **Visualise** action allows you to preview the page, even when it’s not published yet.



1.2.8 Publish your page

When you're happy with the content, you can publish the page and save it.

Have a look at the wonderful work you've just done:



CHAPTER 2

Understanding Novius OS

- Understanding the applications infographics
- Glossary, learn Novius OS jargon

CHAPTER 3

Technical documentation

3.1 Coding standards

These standards for code formatting must be followed by anyone contributing to Novius OS.

You can use the ruleset.xml for PHP_CodeSniffer made for Novius OS.

3.1.1 Case

All keywords are in lowercase (class, interface, extends, implements, abstract, final, var, const, function, public, private, protected, static, if, else, elseif, foreach, for, do, switch, while, try, catch, true, false and null).

All constant names are in uppercase (global or class constant).

3.1.2 Signature of control statements

```
try {
    ...
} catch (...) {
    ...
}

do {
    ...
} while (...);

while (...) {
    ...
}

// A single space between each condition in 'for' loops.
for ($i = 0; $i < 10; $i++) {
    ...
}

// A single space between each condition in 'foreach' loops.
foreach ($array as $key => $item) {
```

```
    ...
}

if (...) {
    ...
} else if (...) {
    ...
} else {
    ...
}

// A single space after cast tokens.
$variable = (array) $variable;
```

3.1.3 Function declaration

```
/*
The opening brace of a function is on the line after the function declaration.
No space before comma.
A single space after comma.
A single space between type hint and argument.
A single space before '=' sign of default value.
A single space after '=' sign of default value.
Parameters with a default value come at the end of the function signature.
*/
function myfunction(array $first, $second, $third = array())
{
        ...
}
```

3.1.4 Function call

```
/*
No space before comma.
A single space after comma.
*/
$value = myfunction($first, $second, $third);
```

3.1.5 Class

```
/*
The opening brace of a class must be on the line after the definition.
There must be one blank line after the namespace declaration.
All class members have scope modifiers (variables, functions and methods).
A single space after scope keywords (private, public, protected, static).
*/
namespace Name\Space;

class MyClass
{
        private $variable1 = null;

        protected static $variable1 = true;
```

```
        const MY_CONSTANT = false;

        public static function myfunction()
        {
                ...
        }
}
```

3.1.6 File

End of line characters must be n. Files do not end with a closing tag.

Only one statement by line.

PHP code must use the long <?php ?> tags or the short-echo <?= ?> tags; it must not use the other tag variations.

Closing braces of scopes correctly aligned. Control structures are correctly indented (4 spaces, no tab).

No additional white space at start and end of file.

3.2 Applications

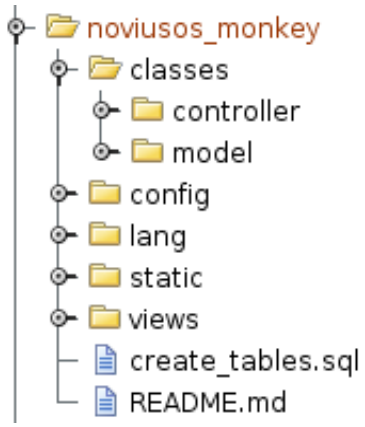
Applications are the easiest way to add functionalities to Novius OS. A few applications are available : Blog, News stories, Slideshows... and Monkeys, a sample application.

The application management system has been implemented so that is is really easy to add and extend applications.

- *Files organisation*
- *Metadata configuration file*
- *App Desk*

3.2.1 Files organisation

The files are organised the same way as application in FuelPHP.



3.2.2 Metadata configuration file

The metadata configuration file is mandatory. It indicates to the core the key information it needs to use the application. Below is the metadata file of the Monkeys application:

```
<?php
return array(
        'name'    => 'Monkey : Novius OS Application Bootstrap',
        'version' => '0.1',
        'icon16' => 'static/apps/noviusos_monkey/img/16/monkey.png',
        'provider' => array(
                'name' => 'Provider',
        ),
        'namespace' => 'Nos\Monkey',
        'launchers' => array(
                'provider_launcher' => array(
                        'name'    => 'Monkey',
                        'url' => 'admin/noviusos_monkey/appdesk',
                        'iconUrl' => 'static/apps/noviusos_monkey/img/32/monkey.png',
                        'icon64'  => 'static/apps/noviusos_monkey/img/64/monkey.png',
                ),
        ),
        'enhancers' => array(
                'noviusos_monkey' => array(
                        'title' => 'Monkey',
                        'desc'  => '',
                        //'enhancer' => 'noviusos_monkey/front',
                        'urlEnhancer' => 'noviusos_monkey/front/main',
                        'iconUrl' => 'static/apps/noviusos_monkey/img/16/monkey.png',
                        'previewUrl' => 'admin/noviusos_monkey/application/preview',
                        'dialog' => array(
                                'contentUrl' => 'admin/noviusos_monkey/application/popup',
                                'width' => 450,
                                'height' => 200,
                                'ajax' => true,
                        ),
                ),
        ),
);
```

The metadata configuration file returns a key => value array:

- `name`: name of your application
- `version`
- `icon16`: 16x16 icon representing your application (used for tabs)
- `provider`: key => value array giving informations about the provider
- `namespace`: namespace of all classes in your application
- `launchers`: key => value array providing informations about launchers (icons at desktop which opens your application in a new tab)

  The `key` is the launchers identifier.

  Value is key => value array.

  - `name`
  - `url`: url called when opening your application

  - `iconUrl`: 32x32 icon for the tab that open your application
  - `icon64`: 64x64 icon for the desktop
- `enhancers`: see *Enhancers*
- `data-catchers`: see *Sharing*
- `template`: key => value array giving registered templates

```
<?php
        'templates' => array(
                'top_menu' => array(
                        'file' => 'noviusos_templates_basic::top_menu',
                        'title' => 'Default template with a top menu',
                        'cols' => 1,
                        'rows' => 1,
                        'layout' => array(
                                'content' => '0,0,1,1',
                        ),
                ),
                // ...
        ),
```

Each templates can be seperated in different parts. You can have a standard template where all is displayed in one location, but you can also have more complexe templates with a left and a right side for example. The idea is to provide this information to Novius OS in order to allow the core to manage multiple wysiwygs. Wysiwygs are displayed in a grid: you are able to choose position and scale of their wysiwygs.

- `file`: location of the template
- `title`: title of the template, will be used when you create / edit a page and choose a template for it.
- `cols`: number of columns in the grid
- `rows`: number of rows in the grid
- `layout`: layout of the wysiwygs in the grid. key => value array
  - the key is the template identifier
  - the value is a string, representing the left, top position, and width, height, comma seperated

3.2.3 App Desk

The App Desk allows you to easily list and filter your application data. See *App Desk*.

3.3 App Desk

The App Desk is the standard UI for applications' home. It includes a main grid, surrounded by inspectors. The idea is to easily make available a central listing and filtering system to developers.

- *General informations*
- *Main configuration*
- *Inspectors configuration*
  - *Model inspector*
  - *Tree model inspector*

- *Date inspector*
- *Data inspector*

3.3.1 General informations

When the user opens the App Desk, the main grid and the inspectors are loaded only once and retrieve data through json ajax loading.

- The main grid calls a controller which extends `\Nos\Controller_Admin_Appdesk`
- The inspectors call a controller which generally extends one of the standard inspectors controller :
  - `\Nos\Controller_Inspector_Model`: standard model filtering
  - `\Nos\Controller_Inspector_Modeltree`: tree organized model filtering
  - `\Nos\Controller_Inspector_Date`: date filtering
  - `\Nos\Controller_Inspector_Data` : static filtering

App Desk and inspectors were coded in Novius OS with the philosophy that everything had to be done through the configurations files. Therefore, if we take a look at the Monkey controller which extends `\Nos\Controller_Admin_Appdesk`:

```
namespace Nos\Monkey;

class Controller_Admin_Appdesk extends \Nos\Controller_Admin_Appdesk
{
}
```

The controller doesn't implement any functions. Everything is loaded in the configuration file associated by default to this controller. See below this configuration file in our Monkey application:

```
<?php
use Nos\I18n;

I18n::load('noviusos_monkey::item');

return array(
        'query' => array(
                'model' => 'Nos\Monkey\Model_Monkey',
                'related' => array('species'),
                'order_by' => array('monk_name' => 'ASC'),
                'limit' => 20,
        ),
        'search_text' => 'monk_name',
        'selectedView' => 'default',
        'views' => array(
                'default' => array(
                        'name' => __('Default view'),
                ),
        ),
        'dataset' => array(
                'id' => 'monk_id',
                'name' => 'monk_name',
                'species' => array(
                        'value' => function($item) {
                                return $item->species->mksp_title;
                        },
```

```
            ),
            'url' => array(
                    'value' => function($item) {
                            return $item->url_canonical(array('preview' => true));
                    },
            ),
            'actions' => array(
                    'visualise' => function($item) {
                            $url = $item->url_canonical(array('preview' => true));

                            return !empty($url);
                    }
            ),
    ),
    'inputs' => array(
            'monk_species_id' => function($value, $query) {
                    if ( is_array($value) && count($value) && $value[0]) {
                            $query->where(array('monk_species_id', 'in', $value));
                    }

                    return $query;
            },
    ),
    'appdesk' => array(
            'tab' => array(
                    'label' => __('Monkey'),
                    'iconUrl' => 'static/apps/noviusos_monkey/img/32/monkey.png'
            ),
            'actions' => array(
                    'update' => array(
                            'action' => array(
                                    'action' => 'nosTabs',
                                    'tab' => array(
                                            'url' => "admin/noviusos_monkey/monkey/insert_update/
                                            'label' => __('Edit'),
                                    ),
                            ),
                            'label' => __('Edit'),
                            'primary' => true,
                            'icon' => 'pencil'
                    ),
                    'delete' => array(
                            'action' => array(
                                    'action' => 'confirmationDialog',
                                    'dialog' => array(
                                            'contentUrl' => 'admin/noviusos_monkey/monkey/delete/
                                            'title' => __('Delete a monkey'),
                                    ),
                            ),
                            'label' => __('Delete'),
                            'primary' => true,
                            'icon' => 'trash'
                    ),
                    'visualise' => array(
                            'label' => 'Visualise',
                            'primary' => true,
                            'iconClasses' => 'nos-icon16 nos-icon16-eye',
                            'action' => array(
```

```
                        'action' => 'window.open',
                        'url' => '{{url}}?_preview=1'
                    ),
                ),
            ),
            'reloadEvent' => 'Nos\\Monkey\\Model_Monkey',
            'appdesk' => array(
                'buttons' => array(
                    'monkey' => array(
                        'label' => __('Add a monkey'),
                        'action' => array(
                            'action' => 'nosTabs',
                            'method' => 'add',
                            'tab' => array(
                                'url' => 'admin/noviusos_monkey/monkey/insert
                                'label' => __('Add a new monkey'),
                            ),
                        ),
                    ),
                    'species' => array(
                        'label' => __('Add a species'),
                        'action' => array(
                            'action' => 'nosTabs',
                            'method' => 'add',
                            'tab' => array(
                                'url' => 'admin/noviusos_monkey/species/inse
                                'label' => 'Add a species'
                            ),
                        ),
                    ),
                ),
                'splittersVertical' => 250,
                'grid' => array(
                    'urlJson' => 'admin/noviusos_monkey/appdesk/json',
                    'columns' => array(
                        'name' => array(
                            'headerText' => __('Name'),
                            'dataKey' => 'name'
                        ),
                        'lang' => array(
                            'lang' => true
                        ),
                        'species' => array(
                            'headerText' => __('Species'),
                            'dataKey' => 'species'
                        ),
                        'published' => array(
                            'headerText' => __('Status'),
                            'dataKey' => 'publication_status'
                        ),
                        'actions' => array(
                            'actions' => array('update', 'delete', 'visualise'),
                        ),
                    ),
                ),
                'inspectors' => array(
                    'speciess' => array(
                        'reloadEvent' => 'Nos\\Monkey\\Model_Species',
```

```
                                        'label' => __('Speciess'),
                                        'url' => 'admin/noviusos_monkey/inspector/species/list',
                                        'grid' => array(
                                                'columns' => array(
                                                        'title' => array(
                                                                'headerText' => __('Species'),
                                                                'dataKey' => 'title'
                                                        ),
                                                        'actions' => array(
                                                                'showOnlyArrow' => true,
                                                                'actions' => array(
                                                                        array(
                                                                                'action' => array(
                                                                                        'action' => '
                                                                                        'tab' => arra
                                                                                                'url'
                                                                                                'labe
                                                                                        ),
                                                                                ),
                                                                                'label' => __('Edit')
                                                                                'name' => 'edit',
                                                                                'primary' => true,
                                                                                'icon' => 'pencil'
                                                                        ),
                                                                        array(
                                                                                'action' => array(
                                                                                        'action' => '
                                                                                        'dialog' => a
                                                                                                'cont
                                                                                                'titl
                                                                                        ),
                                                                                ),
                                                                                'label' => __('Delete
                                                                                'name' => 'delete',
                                                                                'primary' => true,
                                                                                'icon' => 'trash'
                                                                        ),
                                                                ),
                                                        ),

                                                ),
                                                'urlJson' => 'admin/noviusos_monkey/inspector/species
                                        ),
                                        'inputName' => 'monk_species_id[]',
                                        'vertical' => true,
                                ),
                        ),
                ),
        ),
);
```

3.3.2 Main configuration

The main grid configuration is defined through several keys:

- `query`: parameters of the query executed when loading the data. This parameters are the same as the static `find` function in the orm.

```php
<?php
        'query' => array(
                'model' => 'Nos\Monkey\Model_Monkey',
                'related' => array('species'),
                'order_by' => array('monk_name' => 'ASC'),
                'limit' => 20,
        ),
```

- `search_text`: column(s) to search into when the search input is filled in the main grid. It can be an `array` if search in multiple columns or a `string` if single.

```php
<?php
        'search_text' => 'monk_name',
```

- `views`: array key => values. Differents views available

```php
<?php
        'views' => array(
                'default' => array(
                        'name' => __('Default view'),
                        'json' => array(
                                'static/novius-os/admin/config/media/common.js',
                                'static/novius-os/admin/config/media/media.js'
                        ),
                ),
                // ...
        ),
```

  - `name`: name of the view
  - `json`: json sources

- `selectedView`: view by default

```php
<?php
        'selectedView' => 'default'
```

- `dataset`: key => value hash returned via ajax.

```php
<?php
        'dataset' => array(
                'id' => 'monk_id',
                'name' => 'monk_name',
                'species' => array(
                        'value' => function($item) {
                                return $item->species->mksp_title;
                        },
                ),
                'url' => array(
                        'value' => function($item) {
                                return $item->url_canonical(array('preview' => true));
                        },
                ),
                'actions' => array(
                        'visualise' => function($item) {
                                $url = $item->url_canonical(array('preview' => true));

                                return !empty($url);
                        }
```

```
                ),
        ),
```

  - if the value is a string, then its value is the column of the object (for example `monk_id` will get `$monkey->monk_id`)
  - if the value is an array
    * the `value` key is a callback function which allows you to customize the value (for example take a look at `species` key which return the name of the monkey species)
    * if key is `actions`, then the keys define whether or not the action are enabled (for example, the callback of `visualise` return true if the `visualise` action is enabled, false otherwise)

- `inputs`: is a key => value array allowing to apply filtering on the list (requested by the inspectors)

```
<?php
        'inputs' => array(
                'monk_species_id' => function($value, $query) {
                        if ( is_array($value) && count($value) && $value[0]) {
                                                        $query->where(array('monk_species_id
                        }

                        return $query;
                },
        ),
```

  - the key is the input name
  - the value is a callback function with two parameters
    * the first parameter is the value of the input
    * the second parameter is the query : the callback function have to modify the query object in order to apply the filtering requested

All keys we have enumerated since now have an effect on the json result returned by the controller. We will now get into the `appdesk` key, which determine the display of grid : columns title, actions display, inspector display and positions...

The `appdesk` key defines a key => value array:

- `tab`: how the tab is represented (same parameter of the tabs in the [[JavaScript API | (EN) JavaScript API]])

```
<?php
        'tab' => array(
                'label' => __('Monkey'),
                'iconUrl' => 'static/apps/noviusos_monkey/img/32/monkey.png'
        ),
```

- `actions`: predefined actions used by the appdesk, key => value array:

```
<?php
        'actions' => array(
                'update' => array(
                        'action' => array(
                                'action' => 'nosTabs',
                                'tab' => array(
                                        'url' => "admin/noviusos_monkey/monkey/insert_update
                                        'label' => __('Edit'),
                                ),
```

```
                        ),
                        'label' => __('Edit'),
                        'primary' => true,
                        'icon' => 'pencil'
                    ),
                    //...
            ),
```

  - the key is the action name
  - value, key => value array:
    * `action`: define the action executed when the action button is clicked. The parameters inside are the same as in [[JavaScript actions | (EN) JavaScript actions]]
    * `label`
    * `primary`:
      · if set to true, the button will always be shown as standalone
      · if set to false, if there is more than two secondary button, the action will appear inside the drop down button ; otherwise it will also appear as a standalone button
    * `iconClasses`: set the css classes of the button's icon
    * `icon`: shorcut for iconClasses. Will set the css class of the button's icon to the jquery ui css class (for example, if the value is `pencil`, then the css classes will be `ui-icon ui-icon-pencil`)

- `reloadEvent`: the appdesk listens to the associated event (events if the value is an array). Take a look at events in the *JavaScript API*

```
<?php
        'reloadEvent' => 'Nos\Monkey\Model_Monkey',
```

- `appdesk`: display of the appdesk, key => value array:
  - `buttons`: upper buttons that are generally intended to add objects. It is a key => value array

```
<?php
        'buttons' => array(
                'monkey' => array(
                        'label' => __('Add a monkey'),
                        'action' => array(
                                'action' => 'nosTabs',
                                'method' => 'add',
                                'tab' => array(
                                        'url' => 'admin/noviusos_monkey/monkey/insert_up
                                        'label' => __('Add a new monkey'),
                                ),
                        ),
                ),
                //...
        ),
```

    * the key is the name of the action
    * the value is a key => value array:
      · `label`

        - `action`: define the action executed when the button is clicked. The parameters inside are the same as in [[JavaScript actions | (EN) JavaScript actions]]

- `splittersVertical`: position of the vertical splitter (distance from the left border in pixel)

```
<?php
        'splittersVertical' => 250,
```

- `grid`: display of the main grid

```
<?php
        'grid' => array(
                'urlJson' => 'admin/noviusos_monkey/appdesk/json',
                'columns' => array(
                        'name' => array(
                                'headerText' => __('Name'),
                                'dataKey' => 'name'
                        ),
                        'lang' => array(
                                'lang' => true
                        ),
                        // ...
                        'actions' => array(
                                'actions' => array('update', 'delete', 'visualise'),
                        ),
                ),
        ),
```

- `urlJson`: url called to load via ajax the json data
- `columns`: key => value array which defines how the columns are displayed
    * `headerText`: head title of the column
    * `dataKey`: key of an item of the data received
    * `lang`: languages of the item if it has the translatable behaviour
    * `actions`: actions buttons, for each element of the array:
        - if it is a string, then it comes from the key related predefined action
        - it it is an array, it is a custom action which is defined the same way as the predefined actions
- `inspectors`: key => value array which define the inspectors

```
<?php
        'inspectors' => array(
                'speciess' => array(
                        'reloadEvent' => 'Nos\Monkey\Model_Species',
                        'label' => __('Speciess'),
                        'url' => 'admin/noviusos_monkey/inspector/species/list',
                        'inputName' => 'monk_species_id[]',
                        'vertical' => true,
                        'grid' => array(
                                'columns' => array(
                                        'title' => array(
                                                'headerText' => __('Species'),
                                                'dataKey' => 'title'
                                        ),
                                        'actions' => array(/* ... */),
```

```
                                        ),
                                        'urlJson' => 'admin/noviusos_monkey/inspector/species/js
                                ),
                        ),
                ),
```

* the key is equivalent to the inspector name
* `reloadEvent`: the event name that will trigger the inspector reload
* `label`: title label of the inspector
* `url`: url of the html structure of the inspector (loaded at the begining)
* `inputName`: filter name affected by the inspector
* `vertical`: if true, the inspector will be on the left, otherwise if will be on the top
* `grid`: same as the main grid except it is for the inspectors

3.3.3 Inspectors configuration

Model inspector

Example of configuration:

```
<?php
return array(
        'query' => array(
                'model' => 'Nos\Monkey\Model_Species',
                'order_by' => array('mksp_title' => 'ASC'),
        ),
        'dataset' => array(
                'id' => 'mksp_id',
                'title' => 'mksp_title',
        ),
);
```

The configuration has two keys :

- `query`: which defines the query executed for retrieving the data:
  - `model` is the model's class
  - all other columns are used for the query
- `dataset`: key => value hash returned via ajax, same as in the appdesk configuration

Tree model inspector

Example of configuration:

```
<?php
return array(
        'models' => array(
                array(
                        'model' => 'Nos\BlogNews\Blog\Model_Category',
                        'order_by' => 'cat_sort',
                        'childs' => array('Nos\BlogNews\Blog\Model_Category'),
                        'dataset' => array(
```

```
                                'id' => 'cat_id',
                                'title' => 'cat_title',

                        ),
                ),
        ),
        'roots' => array(
                array(
                        'model' => 'Nos\BlogNews\Blog\Model_Category',
                        'where' => array(array('cat_parent_id', 'IS', \DB::expr('NULL'))),
                        'order_by' => 'cat_sort',
                ),
        ),
);
```

- `models`:
  - `model`: model's class
  - `childs`: children's classes
  - `dataset`: same as in the appdesk configuration
  - other columns can be applied to the query object
- `roots`: how to load the root nodes of the tree
  - `model`: model's class
  - other columns can be applied to the query object

Date inspector

```
<?php
return array(

        'input_begin'           => 'date_begin',
        'input_end'             => 'date_end',
        'label_custom'          => 'Custom dates',
        'label_custom_inputs'   => 'from xxxbeginxxx to xxxendxxx',
        'options'               => array('custom', 'since', 'month', 'year'),
        'since'                 => array(
                'optgroup'  => 'Since',
                'options'   => array(
                        '-3 day'            => '3 last days',
                        'previous monday'   => 'Week beginning',
                        '-1 week'           => 'Less than a week',
                        'current month'     => 'Month beginning',
                        '-1 month'          => 'Less than one month',
                        '-2 month'          => 'Less than two months',
                        '-3 month'          => 'Less than three months',
                        '-6 month'          => 'Less than six months',
                        '-1 year'           => 'Less than one year',
                ),
        ),
        'month'                 => array(
                'optgroup'      => 'Previous months',
                'first_month'   => 'now',
                'limit_type'    => 'year',
                'limit_value'   => 1,
```

```
        ),
        'year'                  => array(
                'optgroup'      => 'Years',
                'first_year'    => 'now',
                'limit'         => 4,
        ),
);
```

Data inspector

Example of configuration:

```
<?php
return array(
        'data' => array(
                array(
                        'id' => 'image',
                        'title' => 'Images',
                        'icon' => 'image.png',
                ),
                array(
                        'id' => 'document',
                        'title' => 'Documents',
                        'icon' => 'document-office.png',
                ),
/* ... */
```

The `data` is simply sent via json. Each element in data has:

- `id`
- `title`
- `icon`: which is displayed at the left of the title (optionnal)

3.4 Behaviours

`Behaviours` are meant to extend `Orm\Model` features in a re-usable way.

They are similar to Observers but more powerful.

Like `Observers`, they have options to be configured.

Like `Observers`, they can catch events to act on a model (for example, `before_save`).

In addition, they also provide a set of new methods on the instantiated `Model` item. Sometimes they also provide new events.

- *Publishable*
- *Sortable*
- *Translatable*
- *Tree*
- *Urlenhancer*
- *Virtual name*

- *Virtual path*
- Sharable: see *Sharing and data catchers*

Printer friendly

3.4.1 Publishable

Note: only a yes/no state is supported for now. Full publication start & end date will come later.

1. Column

- `publication_bool_property [bool]`: which column is used to store the state of publication (yes/no)

2. Configuration

The publishable behaviour doesn't have options.

3. Methods provided on an item

`(bool) published()`

Returns whether the item is published or not

4. Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Publishable' => array(
                        'publication_bool_property' => 'page_published',
                ),
        );
}

$page = Model_Page::find(1);

if ($page->published()) {
        // Do something
}
```

3.4.2 Sortable

1. Column

- `sort_property [double]`: which column is used to store the sort order / rank

2. Configuration

- `sort_order`: ASC (default) or DESC

3. Methods provided on an item

`move_before($item)`

`move_after($item)`

`move_to_last_position()`

4. Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Sortable' => array(
                        'events' => array('after_sort', 'before_insert'),
                        'sort_property' => 'page_sort',
                ),
        );
}

$page_1 = Model_Page::find(1);
$page_2 = Model_Page::find(2);

$page_2->move_after($page_1);
```

3.4.3 Translatable

This behaviour needs 3 columns in the table to operate properly.

1. Columns

- `common_id_property [int]`: which column is used to store the common ID
- `single_id_property [int]`: which column is used to store the single ID (equals to common ID for the main lang, `null` otherwise)
- `lang_property [varchar(5)]`: which column is used to store the locale code (examples: `en_GB` or `fr_FR`)

2. Configuration

- `default_lang`: default lang to use when not specified (for example upon creation)

3. Methods provided on an item

get_lang()

Returns the current lang of the item.

is_main_lang()

Returns true or false.

find_main_lang()

Returns the item in the main lang, `null` otherwise.

find_lang($lang)

Possible values for `$lang`:

- `lang_COUNTRY`: returns the item in the appropriate language, if it exists, `null` otherwise.
- `all`: returns an array of all versions of this item.
- `main`: returns the item in the main lang. Alias for `find_main_lang()`.

get_all_lang()

Returns an array of all the langs of the item. Key is the item ID. Value is the locale code.

```
<?php array(
        2 => 'en_GB',
        5 => 'fr_FR',
);
```

get_other_lang()

Same as `get_all_lang()`, but it doesn't include the current item lang.

4. Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Translatable' => array(
                        'events' => array('before_insert', 'after_insert', 'before_save', 'after_dele
                        'lang_property'      => 'page_lang',
                        'common_id_property' => 'page_lang_common_id',
                        'single_id_property' => 'page_lang_single_id',
                ),
        );
}
```

3.4.4 Tree

1. Columns

- `level_property [int]`: optional. Which column is used to store nesting level (depth).

2. Configuration

- `parent_relation`: relation used to fetch the parent.
- `children_relation`: relation used to fetch the children.

3. Methods provided on an item

`get_parent()`

Returns the parent of this item, if it exists, `null` otherwise.

`set_parent($new_parent)`

Sets a new parent for the item.

Can throw an Exception if the item is being moved inside its own sub-tree.

If the item is `Translatable` and exists in multiple languages, all languages will be moved synchronously. This can throw an Exception if the new parent doesn't exists in one of the languages of the moved item.

`find_children($where = array(), $order_by = array(), $options = array())`

Returns all direct children of the item, optionally filtered and ordered by the parameters.

The method uses native's Fuel `find()` method, passing along the `$options` parameters as follow:

```
<?php
$options = \Arr::merge($options, array(
        'where'    => $where,
        'order_by' => $order_by,
));
```

`find_children_recursive($include_self)`

Returns all children and sub_children of the item.

- `$include_self`: Should the returned array contain the item itself?

`find_root()`

Returns the highest parent in the tree, or `null` if the item has no parent.

4. Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Tree' => array(
                        'events' => array('before_query', 'after_delete'),
                        'parent_relation' => 'parent',
                        'children_relation' => 'children',
                        'level_property' => 'page_level',
                ),
        );

        protected static $_has_many = array(
                'children' => array(
                        'key_from'       => 'page_id',
                        'model_to'       => 'Nos\Model_Page',
                        'key_to'         => 'page_parent_id',
                        'cascade_save'   => false,
                        'cascade_delete' => false,
                ),
        );

        protected static $_belongs_to = array(
                'parent' => array(
                        'key_from'       => 'page_parent_id',
                        'model_to'       => 'Nos\Model_Page',
                        'key_to'         => 'page_id',
                        'cascade_save'   => false,
                        'cascade_delete' => false,
                ),
        );

}
```

3.4.5 Urlenhancer

This behaviour provides helper URL methods relative to models which are displayed by *URL Enhancers*.

1. Column

This behaviour doesn't need columns.

2. Configuration

- `enhancers [array of strings]`: array of enhancer names able to generate an URL for this item.

The enhancers listed here must define a `get_url_model($item, $params)` method. See the *related documentation* for more details.

3. Methods provided on an item

All 3 methods take the same `$params` array:

- `preview`: true or false. Whether to include unpublished pages. You have to include `?_preview=1` yourself.

`urls($params = array())`

Returns an array of all available URL for this item. The array contains:

```
<?php
array(
        'page_id::item_slug' => 'full_url (relative to base)',
);
```

This way, we get all the informations we want:

- The page ID
- The generated URL by the enhancer (item slug)
- The current full URL (current page URL + enhancer part)

If there's no result, the function will return empty `array()`

`url_canonical($params = array())`

Apply only to items which have the `Sharable` behaviour. Returns the URL configured manually in the shared data (content nugget).

If the item is not shareable or has no URL configured manually, it's equivalent to `$item->url();`

`url($params = array())`

Returns a valid URL for the item, or `null` of the item is not displayed.

4. Example

```
<?php
class Model_Monkey extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Urlenhancer' => array(
                        'enhancers' => array('noviusos_monkey'),
                ),
        );
}
```

3.4.6 Virtual name

This will use the `title_property` of the model to auto-generate the virtual name (slug).

Upon `save()`, an Exception will be throwed for item that does not fulfill the `unique` requirement.

1. Column

- `virtual_name_property [varchar]`: which column is used to store the virtual name.

2. Configuration

- `unique`: true / false or 'lang'

3. Methods provided on an item

`virtual_name()`

Returns the virtual name of the item.

4. Example

```
<?php
class Model_Monkey extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Virtualname' => array(
                        'events' => array('before_save', 'after_save'),
                        'virtual_name_property' => 'monk_virtual_name',
                ),
        );
}
```

3.4.7 Virtual path

This is an extension of the `Virtual name` behaviour.

1. Columns

- `virtual_name_property [varchar]`: inherited from *Virtual name*
- `virtual_path_property [varchar]`: which column is used to store the virtual path.

2. Configuration

- `unique`: inherited from `Virtual name`
- `extension[before]`: String to use to prepend extension
- `extension[after]`: String to use to append extension
- `extension[property]`: column to use to fetch the extension of the virtual path (slug).
- `parent_relation`: Relation used to generate the first part of the virtual path (slug).

3. Methods provided on an item

`virtual_path()`

Returns the virtual path of the item.

4. Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Virtualpath' => array(
                        'events' => array('before_save', 'after_save', 'change_parent'),
                        'virtual_name_property' => 'page_virtual_name',
                        'virtual_path_property' => 'page_virtual_url',
                        'extension_property' => '.html',
                        'parent_relation' => 'parent',
                ),
        );
}
```

3.4.8 Printer friendly

Publishable

- publication_bool_property [bool]
-
- (bool) published()

Sortable

- sort_property [double]
- sort_order
-
- move_before($item)
- move_after($item)
- move_to_last_position()

Translatable

- common_id_property [int]
- single_id_property [int]
- lang_property [varchar(5)]
- default_lang
-
- get_lang()
- is_main_lang()
- find_main_lang()

- `find_lang($lang)`: `lang_COUNTRY`, 'all' or 'main'
- `get_all_lang()`
- `get_other_lang()`

Tree

- `parent_relation`
- `children_relation`
- `level_property`
-
- `get_parent()`
- `set_parent($new_parent)`
- `find_children($where = array(), $order_by = array(), $options = array())`
- `find_children_recursive($include_self)`
- `find_root()`

Urlenhancer

- `enhancers[name]`
-
- `urls($params = array())`
- `url_canonical($params = array())`
- `url($params = array())`

Virtual name

- `virtual_name_property [varchar]`
- `unique`: true / false or 'lang'
-
- `virtual_name()`

Virtual path

- `virtual_name_property [varchar]`
- `virtual_path_property [varchar]`
- `unique`: inherited from `Virtual name`
- `extension[before]`: String to use to prepend extension
- `extension[after]`: String to use to append extension
- `extension[property]`: column to use to fetch the extension of the virtual path (slug).
- `parent_relation`: Relation used to generate the first part of the virtual path (slug).

-
- `virtual_path()`

3.5 CRUD controller

The CRUD controller is in charge of generating the forms (add, edit & delete) related to an item / model and handling automatically the multilingual / translation problematic.

3.5.1 Configuration

The configuration file looks like this:

```
<?php
return array(
        'controller_url'  => 'admin/noviusos_monkey/monkey',
        'model' => 'Nos\\Monkey\\Model_Monkey',
        'messages' => array(),
        'tab' => array(),
        'layout' => array(),
        'fields' => array(),
);
```

The `controller_url` key should tell where your controller is located. Here it's in the `noviusos_monkey` application / namespace, and it's name is `Controller_Admin_Monkey`.

The `model` key should tell which model it generates forms for.

The `messages` key contains all generic messages used by CRUD controller.

The `tab` key contains the tab information, such as the icon & label.

The `layout` key tells how the form looks like and where fields are displayed inside it.

The `fields` key contains the fields definition, including:

- a label ;
- how the field is displayed: a native HTML `<input>` or a custom widget (like date picker or wysiwyg) ;
- validation rules.

3.5.2 Jump to

- *Messages list*
- *Tab information*
- ***Layout***
  - *1. Layout standard*
  - *2. Expander*
  - *3. Fields list*
  - *4. Accordion*
- *Fields*

3.5.3 Messages list

This list may not be up to date. Be sure to check the config file included in the sample application for the latest version.

```
<?php
return array(
        'messages' => array(
                'successfully added' => __('Monkey successfully added.'),
                'successfully saved' => __('Monkey successfully saved.'),
                'successfully deleted' => __('The monkey has successfully been deleted!'),
                'you are about to delete, confim' => __('You are about to delete the monkey <span sty
                'you are about to delete' => __('You are about to delete the monkey <span style="font
                'exists in multiple lang' => __('This monkey exists in <strong>{count} languages</str
                'delete in the following languages' => __('Delete this monkey in the following langua
                'item deleted' => __('This monkey has been deleted.'),
                'not found' => __('Monkey not found'),
                'error added in lang' => __('This monkey cannot be added {lang}.'),
                'item inexistent in lang yet' => __('This monkey has not been added in {lang} yet.'),
                'add a item in lang' => __('Add a new monkey in {lang}'),
        )
);
```

3.5.4 Tab information

```
<?php
return array(
        'tab' => array(
                'iconUrl' => 'static/apps/noviusos_monkey/img/16/monkey.png',
                // Add form will user 'insert'
                // Edit form will use item's title
                // Translate form (multilingual) will use 'blank_slate'
                'labels' => array(
                        'insert' => __('Add a monkey'),
                        'blankSlate' => __('Translate a monkey'),
                ),
        ),
);
```

3.5.5 Layout

The layout list all views needed to render the form. The generic format for using a layout is the following:

```
<?php
return array(
        'layout' => array(
                'view_1' => array(
                        'view' => 'nos::form/layout_standard',
                        'params' => array(
                                // View params. Depends on the view.
                        ),
                ),
                // More views can be used here.
        ),
);
```

In addition to view specific params, Novius OS always include the following vars:

- `$item` : the instance of the model currently edited (or added / translated).
- `$fieldset` : the form instance which holds all fields definition.

Configuration shortcut

Because 95% of the time, we want to use `nos::form/layout_standard` as view for the layout, a shortcut was created for simplicity: only write the view `params` of the standard layout.

```
<?php
// The following...
return array(
        'layout' => array(
                'view_1' => array(
                        'view' => 'nos::form/layout_standard',
                        'params' => array(
                                // View params. Depends on the view.
                        ),
                ),
                // More views
        ),
);


// ... is the same as this:
return array(
        'layout' => array(
                // View params for ``nos::form/layout_standard``.
        ),
);
```

It's much more limitating because you can only use one view to render the layout, and it has to be `nos::form/layout_standard`. But that's what should be used 95% of the time.

List of natives views included

Used as container for other layouts / views

- `nos::form/layout_standard`: used as a container for other views ;
- `nos::form/expander`: used inside `layout_standard.content` in the Pages application ;

Used as final view

- `nos::form/fields`: used inside `layout_standard.content` in the User application ;
- `nos::form/accordion`: used inside `layout_standard.menu` in the Pages application.

1. Layout standard

This configuration lists the available params for the `nos::form/layout_standard` view.

```
<?php
array(
        'view' => 'nos::form/layout_standard',
        'params' => array(
                'title' => 'monk_name',
                'medias' => array('medias->thumbnail->medil_media_id'),
                'large' => true,
                'subtitle' => array('monk_species_id'),
                'content' => array(
                        // array of sub-layouts
                ),
                'menu' => array(
                ),
                'save' => 'save',
        )
);
```

2. Expander

This configuration lists the available params for the `nos::form/expander` view.

```
<?php
array(
        'view' => 'nos::form/expander',
        'params' => array(
                'title'   => __('Title'),
                'options' => array(
                        'allowExpand' => false,
                ),
                'content' => array(
                        // array of sub-layouts
                ),
        ),
);
```

3. Fields list

This configuration lists the available params for the `nos::form/fields` view.

```
<?php
array(
        'view' => 'nos::form/fields',
        'params' => array(
                'fields' => array(
                        'monk_summary',
                        'wysiwygs->content->wysiwyg_text',
                ),
                // 'callback' is optional. Default action is shown here
                'callback' => function($field) {
                        echo $field->build();
                },
                // 'begin' is optional. Default is shown here
                'begin' => '<table class="fieldset">',
                // 'end' is optional. Default is shown here
                'begin' => '</table>',
```

```
        ),
);
```

4. Accordion

This configuration lists the available params for the `nos::form/accordion` view.

```
<?php
array(
        'view' => 'nos::form/accordion',
        'params' => array(
                'accordions' => array(
                        'section_1' => array(
                                'title' => __('Section title'),
                                'fields' => array('page_parent_id', 'page_menu', 'page_menu_title'),
                        ),
                        // More sections
                ),
        ),
);
```

3.5.6 Fields

The documentation for this section is the same as in the `form generation` page.

3.6 Enhancers

`Enhancers` are the front side of an application. They have to be added to pages in order to be visible in your website. It is possible to define parameters for customizing the display of an enhancer.

For instance, the blog application have one enhancer available. Lets say you want to display your blog in the home page of your website:

- You edit your home page ;
- On the wysiwyg menu, click "Application", then choose "Blog" ;
- A configuration popup appears (where you can define the number of items you want to display and the category you wish to filter) ;
- Hit "Save" : a preview appears in the wysiwyg ;
- If you visualise the home page on front, the list of existing blog items appears, customised by the parameters you choose earlier.

This documentation will get onto several aspect of the enhancer:

- *1. Define the enhancer in the metadata file*
- *2. [Back-office] Create a configuration popup* (number of item per pages / which categories will be displayed)
- *3. [Back-office] Display a preview in the Wysiwyg*
- *4. [Front-office] Display your content on the website*
- *5. URL enhancers*

3.6.1 1. Define the enhancer in the metadata file

In order to use enhancers you first have to edit the application metadata configuration. All enhancers are defined in the key `enhancers`.

See below as an example the `enhancer` key of the metadata Monkey application.

```
array(
        'enhancers' => array(
                'noviusos_monkey' => array(
                        'title' => 'Monkey',
                        'desc'  => '',
                        'iconUrl' => 'static/apps/noviusos_monkey/img/16/monkey.png',
                        //'enhancer' => 'noviusos_monkey/front',
                        'urlEnhancer' => 'noviusos_monkey/front/main',
                        'previewUrl' => 'admin/noviusos_monkey/application/preview',
                        'dialog' => array(
                                'contentUrl' => 'admin/noviusos_monkey/application/popup',
                                'width' => 450,
                                'height' => 200,
                                'ajax' => true,
                        ),
                ),
        ),
);
```

Each enhancer is represented by a key => configuration pattern. The key is mandatory, configuration must be an array.

Configuration defines main informations about the enhancer:

- `title`: title of the enhancer displayed when opening the Application menu on the page edition wysiwyg.
- `desc`: description of the enhancer
- `iconUrl`: location of the icon displayed when opening the Application menu on the page edition wysiwyg.
- `enhancer` or `urlEnhancer`: url of the application enhancer when displaying it on front. Only one of the two keys can be used : use `enhancer` if you don't want to use url enhancers, `urlEnhancer` otherwise.
- `previewUrl`: url of the enhancer preview in the wysiwyg.
- `dialog`: parameters of the configuration dialog box. Parameters are similar as the ones defined for dialogs in the javascript API.

3.6.2 2. [Back-office] Create a configuration popup

The content of the back-end configuration popup is defined by the `dialog` key. A request is made to `contentUrl` and the content returned is displayed in the configuration popup. In the case of the Monkey application, the popup content loads `admin/noviusos_monkey/application/popup` via ajax.

The popup content is an html form. It is recommended to use the following template for the html form (inspired by the Monkey application) :

```
<div id="<?= $id = uniqid('temp_') ?>">
        <form method="POST" action="admin/noviusos_monkey/application/save">
                <!-- content -->
        </form>
</div>

<script type="text/javascript">
```

```
require([
        'jquery-nos'
        ], function($) {
                $(function() {
                        var div = $('#<?= $id ?>')
                                .find('a[data-id=close]')
                                .click(function(e) {
                                        div.closest('.ui-dialog-content').wijdialog('close');
                                        e.preventDefault();
                                })
                                .end()
                                .find('form')
                                .submit(function() {
                                        var self = this;
                                        $(self).ajaxSubmit({
                                                dataType: 'json',
                                                success: function(json) {
                                                        div.closest('.ui-dialog-content').trigger('sa
                                                },
                                                error: function(error) {
                                                        $.nosNotify('An error occured', 'error');
                                                }
                                        });

                                        return false;
                                })
                                .nosFormUI();
                });
        });
</script>
```

Please note that the form is submited via ajax. The returned content must be a json formatted array with two keys:

- `config` which is the configuration of the enhancer
- `preview` which is an HTML preview of the enhancer

You must then trigger the `save.enhancer` event on the popup content in order to close the popup and display the newly created enhancer in the wysiwyg.

```
div.closest('.ui-dialog-content').trigger('save.enhancer', json);
```

3.6.3 3. [Back-office] Display a preview in the Wysiwyg

The enhancer is displayed on the wysiwyg by loading via ajax the content of `previewUrl`. In the case of the Monkey application, it will display the content returned by `admin/noviusos_monkey/application/preview`.

3.6.4 4. [Front-office] Display your content on the website

Once the wysiwyg is saved and the page published, the enhancer will be available on Front. The content displayed comes from `enhancer` or `urlEnhancer`.

In the case of the Monkey application, the diplayed content is returned from `noviusos_monkey/front/main` hence the `main` action of the `Controller_Front` of the Monkey application.

The action called takes in parameter the configuration of the enhancer (which was set in the configuration popup).

3.6.5 5. URL enhancers

If the key `urlEnhancer` was populated, this will allow the enhancer to manage more complex urls.

Lets say your enhancer is located on "page.html" ; it will then be able to manage urls like "page/1.html", "page/more.html" or "page/more/1.html".

Like in the previous case, the content is still returned by the `main` action, but it is possible to get the extended url by calling the function `$this->main_controller->getEnhancerUrl();` and customize the content.

The controller must in this case implement a `get_url_model()` static function. See below the one in the Monkey application:

```
<?php
public static function get_url_model($item, $params = array())
{
        $model = get_class($item);
        $page = isset($params['page']) ? $params['page'] : 1;

        switch ($model) {
        case 'Nos\Monkey\Model_Monkey' :
                return urlencode($item->monk_virtual_name).'.html';
                break;

        case 'Nos\Monkey\Model_Species' :
                return 'species/'.urlencode($item->mksp_virtual_name).($page > 1 ? '/'.$page : '').'
                break;
        }

        return false;
}
```

This function is related with the Urlenhancer behaviour implemented in the objects (see *Behaviours*). Indeed, `urls($params = array())`, `url_canonical($params = array())` and `url($params = array())` indirectly call the controller static function `get_url_model($item, $params = array())`.

Lets say the monkey enhancer is located on both "first-page.html" and "page-2.html", and we loaded a monkey object `$monkey`. If we call `$monkey->urls();`, the Urlenhancer behaviour will iterate on each published page (of the same language if the object has the translatable behaviour) and complete the URL by calling the `get_url_model()` function (the `$item` parameter will then be `$monkey`, and the `$params` parameter will be the `$params` parameter of the `urls($params = array())` function).

In our case the returned array of `$monkey->urls();` will be similar to:

```
<?php
array(
  '1::monkey-name.html' =>  'first-page/monkey-name.html',
  '2::monkey-name.html' => 'page-2/monkey-name.html'
);
```

3.7 JavaScript API

3.7.1 Summary

- *Notifications*
- *Tabs*

- *Ajax*
- *Dialog*
- *Events*
- *Forms*
- *On Show*

3.7.2 Notifications

Wrapper around the jQuery Pines Notify plugin.

Usage

```
$.nosNotify(message, type);
```

Examples

```
// Simple notification
$.nosNotify('Hello notification');

// Error notification
$.nosNotify('Error message', 'error');
```

3.7.3 Tabs

Usage

```
$(context).nosTabs(action, params)
```

1. add / open

`add` will always create a new tab. `open` will either create a new tab, or refocus an existing tab with the same URL.

The icon size is 16*16px. It can be provided either as:

- an image URL
- CSS classes (background-image / sprite)

Options

```
{
        url: '',
        iframe: false,
        label: '',
        iconClasses: 'ui-icon ui-icon-document',
        iconUrl: ''
}
```

If the URL content returns a valid JSON value, it will be parsed by our native Ajax response handler rather than opening a new tab.

2. close

Closes the current tab.

```
$(context).nosTabs('close');
```

3. update

Update tab informations (URL, label and icon, see `options` from `add / open`).

If an URL is provided, the tab won't be reloaded (replaced) unless you set the `reload: true` option.

Options

- All options from `add` / `open`
- `reload`: false

Example

```
$(context).nosTabs('open', {
        'url' => 'admin/nos/page/page/insert_update/2',
        'label' => "Page's title",
        'iconUrl' => 'static/novius-os/admin/novius-os/img/16/page.png',
});
```

3.7.4 Ajax

Usage

```
$(context).nosAjax(params);
```

The back-office is a big HTML page. Most requests are made using Ajax. Hence Novius OS has is wrapper around jQuery.

The `nosAjax` function has the same API as `$.ajax`. There are two main differences:

- Default options
- Native operations performed on returned JSON

Default options

```
{
        dataType: 'json',
        type: 'POST'
}
```

Callbacks & return value (JSON)

The `success` and `error` callbacks are monkey-patched to perform default native operations (in addition to the callback provided by the user).

The returned JSON support the following actions:

- `notify [string or array]` : displays one or several notification(s).
- `error` [string or array]: displays one or several error notification(s).
- `closeDialog [bool]`: used for ajax requests performed from a dialog.
- `closeTab [bool]`: closes the current tab.
- `replaceTab [string]`: URL of the new tab.
- `dispatchEvent [object or array]`: see `listenEvent()` / `dispatchEvent()`

Example of request

```
// Performs a POST request, expecting JSON response
$.nosAjax({
        'url': 'admin/nos/page/appdesk/clear_cache'
});
```

Example of returned JSON

```
{
        // This will trigger a notification
        'notify': 'Cache has been renewed.'
}
```

3.7.5 Dialog

This API can be used to create a dialog:

- From an exising `<div>`
- From an URL (`<iframe>`)
- From Ajax-fetched content

Default options

```
{
        destroyOnClose : true,
        width: window.innerWidth - 200,
        height: window.innerHeight - 100,
        modal: true,
        title: ''
}
```

Usage

```
// Transform an existing div
$(div).nosDialog();

// Open an iframe
$(context).nosDialog({
        contentUrl: 'http://...'
});

// Load Ajax HTML into the DOM and transform it into a dialog
$(context).nosDialog({
        contentUrl: 'http://...',
        ajax: true,
        // optional data
        ajaxData: {}
});
```

If the Ajax URL returns a valid JSON value, it will be parsed by our native Ajax response handler rather than opening a new tab.

The dialog knows the tab context it's opened from. If you close a tab, any dialog contained within it will appropriately be destoyed.

3.7.6 Events

Some UI widgets are listening to particular events. For example, the appdesk's grid and inspectors are listening to events related to Model they display.

Events are bound to the tab they're being registered from. Events can be dispatched (created) anytime from any tab. Callbacks registered in the active (visible) tab will be triggered instantaneously. Callback registered in non-active tabs will be trigger when it become visible.

Standard event structure used in Novius OS

```
{
        name: '', // model class related to the event. Example: ``Nos\Model_Page``
        id: 1, // primary key (or array) of the model
        action: '', // action performed. Examples: ``insert``, ``update``, ``delete``
        lang: '' // lang (or array) of the model
}
```

Dispatched (created) events usually have a lot of informations than callback listen to. Listeners don't need the whole information to be useful, they can listen to a subset of the full event.

Listening

```
// Page grid listens the following events (if the selected lang is en_GB):
$(context).nosListenEvent([
        {
                name: 'Nos\Model_Page',
                action: ['insert', 'delete']
        },
        {
                name; 'Nos\Model_Page',
```

```
                lang; 'en_GB'
        }
], function() {
        // Reload the grid data
});
```

The context will be listening to:

- all page's creation (from any language) or deletion (any page ID) ;
- any action performed on a page in en_GB (creation, deletion, update).

Both objects will match for the event `{name:'Nos\Model_Page', action:'delete', lang:'en_GB'}`.

Dispatching

This function does not require a context.

```
// This is triggered when a page is inserted
$.nosDispatchEvent({
        name: 'Nos\Model_Page',
        action: 'insert',
        id: 4,
        lang: 'en_GB',
});

// This is triggered when a sub-tree is deleted
$.nosDispatchEvent({
        name: 'Nos\Model_Page',
        action: 'delete',
        id: [4, 5, 9, 8],
        lang: ['en_GB', 'fr_FR'],
});
```

3.7.7 Forms

UI enhancements

`$(context).nosFormUI();` will perform UI enhancements on the form inputs. We rely mainly on Wijmo, which use jQuery UI for the theming:

- `wijtextbox()`
- `wijdropdown()`
- `wijcheckbox()`
- `wijradio()`
- `wijexpander()` for `.expander` elements
- `wijaccordion()` for `.accordion` elements

`<input type="submit">` and `<button>` can have a `data-icon="{icon-name}"` or `data-iconUrl="http://..."` property. `{icon-name}"` is a standard icon name from jQuery UI.

Ajax

`$(context).nosFormAjax();` will use the jquery-form plugin to submit the form using Ajax rather than the native browser action. If there's validation too, it will run before submitting.

Most forms are using it, since it's part of the standard form layout.

Validation

`$(context).nosFormValidate(params);` will use the jquery-validation plugin to perform inline validation on the elements. It's already configured to display error messages nicely, and take into account some specificity from the UI enhancements (like the accordion).

Most forms are using it, since it's part of the standard form layout.

3.7.8 On Show

`nosOnShow()` is a special API which delays the rendering of UI elements when they are visible.

A lot of UI elements don't initialise correctly when they are hidden (they can't calculate sizes properly).

```
// Instead of this:
$(element).widget();

// Do this:
$(element).nosOnShow('one', function() {
        $(this).widget();
});

// You can also use 'bind' instead of 'one': the callback will be executed each time the element is s
```

This way, the element won't be initialised until it becomes visible. If you're responsible for the code displaying an element, then you just need to call `.nosOnShow()` to trigger the delayed actions on the children.

```
// Instead of this:
$(element).show();

// Do this:
$(element).show().nosOnShow();
```

3.8 JavaScript actions

Our JavaScript API provides a centralised way to execute actions without closures.

Those actions are defined only using plain JSON objects (or PHP arrays).

The `$(context).nosAction()` currently works for following actions:

- *nosTabs (wrapper)*
- *nosAjax (wrapper)*
- *window.open*

3.8.1 Generic syntax

```
$(context).nosAction({
        action: 'actionName'
        // other keys depending on the action
}, data);
```

`data` is a JSON object used to replace `{{placeholders}}` in the first parameter.

Placeholders

```
// Define an action using placeholders
var myActionWithPlaceholders = {
        action: 'nosTabs',
        tab: {
                url: 'admin/nos/page/page/insert_update/{{id}}',
                label: '{{title}}',
                iconUrl => 'static/novius-os/admin/novius-os/img/16/page.png'
        }
};

// Define data to fill-in the placehoders
var page = {
        id: 2,
        title: 'Homepage'
};

// Call the action, {{placeholders}} will be replaced before execution
$(context).nosAction(myActionWithPlaceholders, page);


// This is equivalent
var myActionNoPlaceholders = {
        action: 'nosTabs',
        tab: {
                url: 'admin/nos/page/page/insert_update/2',
                label: 'Homepage',
                iconUrl => 'static/novius-os/admin/novius-os/img/16/page.png'
        }
};
$(context).nosAction(myActionNoPlaceholders);
```

3.8.2 nosTabs (wrapper)

Documentation for nosTabs()

Example

```
// 1. Define a JSON action
var myAction = {
        action: 'nosTabs',
        params: {}, // Params, as defined in the nosTabs() API
        method: '' // Optional. Examples: 'add', 'open', 'update', 'close'
};
```

```
// 2. Called it using the nosAction() wrapper
$(context).nosAction(myAction);

// The above is equivalent to
$(context).nosTabs(myAction.method || 'open', myAction.params);
```

3.8.3 nosAjax (wrapper)

Documentation for nosAjax()

Example

```
// 1. Define a JSON action
var myAction = {
        action: 'nosAjax',
        params: {} // Params, as defined in the nosAjax() API
};

// 2. Called it using the nosAction() wrapper
$(context).nosAction(myAction);

// The above is equivalent to
$(context).nosAjax(myAction.params);
```

3.8.4 window.open

Example

```
// 1. Define a JSON action
var myAction = {
        action: 'window.open',
        url: 'http://...'
};

// 2. Called it using the nosAction() wrapper
$(context).nosAction(myAction);

// The above is equivalent to
window.open(myAction.url);
```

3.9 Sharing

3.9.1 Summary

Data catchers are part of the [sharing in Novius OS](http://novius-os.github.com/docs/applications.html#sharing).

> Novius OS is a CMS **built around data.** One of its main roles is to allow the data to flow between applications and in/out of the the OS.

Standardised data types are defined, such as name, URL and image. They are called shared data. Each application indicates the data it releases. The blog application, for instance, provides a name, an URL, an image and a text for every blog post.

Some applications are able to make use of the shared data thanks to their data catchers. These components watch the shared data and interact with the pieces they're able to exploit (e.g. the Twitter application will only need a name and a URL to tweet). Some data catchers require a user action whereas others operate automatically.

3.9.2 Content nuggets: made of shared data

Taxonomy (standardised data type which are part of a content nugget)

- Title
- URL
- Text
- Image

`Nos\Orm_Behaviour_Sharable` (creating content nuggets)

Shared data are defined using the `Sharable` *Behaviour* on a `Model`.

The `data` configuration from the `Sharable` behaviour defines how the default data are generated for the item.

Examples

1. Use a column value as default data

```
<?php
array(
        \Nos\DataCatcher::TYPE_TITLE => array(
                'value' => 'monk_name',
        ),
);
```

2. Use a callback function to compute the default value

```
<?php
array(
        \Nos\DataCatcher::TYPE_TITLE => array(
                'value' => function($monkey) {
                        return $monkey->monk_name;
                },
        ),
);
```

3. Real-world example (found in the Monkey sample application)

```
<?php

class Model_Monkey extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
```

```
                'Nos\Orm_Behaviour_Sharable' => array(
                        'data' => array(
                                \Nos\DataCatcher::TYPE_TITLE => array(
                                        'value' => 'monk_name',
                                        'useTitle' => __('Use monkey name'),
                                ),
                                \Nos\DataCatcher::TYPE_URL => array(
                                        'value' => function($monkey) {
                                                $urls = $monkey->urls();
                                                if (empty($urls)) {
                                                        return null;
                                                }
                                                reset($urls);

                                                return key($urls);
                                        },
                                        'options' => function($monkey) {
                                                return $monkey->urls();
                                        },
                                ),
                                \Nos\DataCatcher::TYPE_TEXT => array(
                                        'value' => function($monkey) {
                                                return $monkey->monk_summary;
                                        },
                                        'useTitle' => __('Use monkey summary'),
                                ),
                                \Nos\DataCatcher::TYPE_IMAGE => array(
                                        'value' => function($monkey) {
                                                $possible = $monkey->possible_medias();

                                                return Arr::get(array_keys($possible), 0, null);
                                        },
                                        'possibles' => function($monkey) {
                                                return $monkey->possible_medias();
                                        },
                                ),
                        ),
                ),
        );
}
```

Some data types, like `url` or `image` have additional parameters.

`options` is used to give a list of all possible values that can be used by the shared data.

For example, if multiple URL are available for a given item, the user will be able to choose which URL he wants to use when sharing the item.

3.9.3 Data catchers (use content nuggets)

Data catchers are components which use content nuggets generated by the models.

Data catchers are defined by applications in the `metadata.config.php` file, the same way as templates, enhancers and launchers.

Included data catchers

Requires user action

- Twitter
- Facebook
- Blog

The Blog data catcher can be used to create new blog posts from other items, such as monkeys (see bootstrap application) or books (this application does not exists, it's just an example).

Operate automatically

- RSS item
- RSS channel

The RSS data catchers operate automatically, but they won't be available for all models, as they require some extra configuration on a per-model & per-enhancer basis. This first draft of the documentation won't cover this functionality.

How the Twitter data catcher is defined

```
<?php
return array(
        'data_catchers' => array(
                'noviusos_twitter_intent' => array(
                        'title' => 'Twitter',
                        'description'  => '',
                        'iconUrl' => 'static/apps/noviusos_twitter/img/twitter.png',
                        // Which action is triggered when the user click on the button
                        // Replacements are made using the shared data (content nugget)
                        'action' => array(
                                'action' => 'window.open',
                                'url' => 'https://twitter.com/intent/tweet?text={{'.\Nos\DataCatcher
                        ),
                        'onDemand' => true, // Requires user action
                        'specified_models' => false, // Apply on every model (with the sharable beha
                        'required_data' => array(
                                \Nos\DataCatcher::TYPE_TITLE,
                        ),
                        'optional_data' => array(
                                \Nos\DataCatcher::TYPE_URL,
                        ),
                ),
        ),
);
```

So the **Twitter** data catcher only requires the content nugget to have a title. The URL is optional (but will be used if provided).

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*