Novius OS Documentation

Release Chiba 1

Novius

November 22, 2013

Contents

Welcome to Novius OS API documentation. It is hosted and generated by Read The Docs.

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

Sources are on Git Hub, we look forward to your `Pull Request`. Contact us if you need help with your contribution.

- English documentation
- Version française
- Japanese version

CHAPTER 1

Summary

1.1 PHP API

1.1.1 Novius OS configuration

Multi-Contexts

Configuration

To change the contexts of your Novius OS instance, edit the `local/config/contexts.config.php` file.

Default configuration After installation, Novius OS is configured with three contexts, one site in three languages:

```php
<?php
return array(
        'sites' => array(
                'main' => array(
                        'title' => 'Main site',
                        'alias' => 'Main',
                ),
        ),

        'locales' => array(
                'fr_FR' => array(
                        'title' => 'Français',
                        'flag' => 'fr',
                ),
                'en_GB' => array(
                        'title' => 'English',
                        'flag' => 'gb',
                ),
                'ja_JP' => array(
                        'title' => '',
                        'flag' => 'jp',
                ),
        ),
        'contexts' => array(
                'main::en_GB' => array(),
                'main::fr_FR' => array(),
                'main::ja_JP' => array(),
        ),
);
```

Multi-sites / multi-languages Here is an example configuration with five contexts across three sites and three languages:

```
<?php
return array(
        'sites' => array(
                'main' => array(
                        'title' => 'Main site',
                        'alias' => 'main',
                ),
                'mobile' => array(
                        'title' => 'Mobile site',
                        'alias' => 'Mobile',
                ),
                'event' => array(
                        'title' => 'Event site',
                        'alias' => 'Event',
                ),
        ),

        'locales' => array(
                'en_GB' => array(
                        'title' => 'English',
                        'flag' => 'gb',
                ),
                'fr_FR' => array(
                        'title' => 'Français',
                        'flag' => 'fr',
                ),
                'es_ES' => array(
                        'title' => 'Español',
                        'flag' => 'es',
                ),
        ),

        'contexts' => array(
                'main::en_GB' => array(),
                'main::fr_FR' => array(),
                'main::es_ES' => array(),
                'mobile::fr_FR' => array(),
                'event::en_GB' => array(),
        ),
);
```

One site in one language Here is an example configuration for just one site in one language:

```
<?php
return array(
        'sites' => array(
                'main' => array(
                        'title' => 'Main site',
                        'alias' => 'main',
                ),
        ),

        'locales' => array(
                'en_GB' => array(
                        'title' => 'English',
                        'flag' => 'gb',
                ),
        ),

        'contexts' => array(
                'main::en_GB' => array(),
        ),
);
```

Domain Names

By default, the first context will answer to the root of your domain, the following contexts in a `site_code/language_code/` subdirectory (e.g.: `main/es_ES/`).

But for each context, you can choose the URI (including a domain and optionally a directory) it will answer to, by specifying it the the associated configuration array.

Contexts on subdirectory

```
    <?php

'contexts' => array(
    'main::en_GB' => array(), // Uses the default domain
    'main::fr_FR' => array(
        'http://www.mysite.com/fr/',
    ),
    'main::es_ES' => array(
        'http://www.mysite.com/es/',
    ),
    'mobile::fr_FR' => array(
        'http://www.mysite.com/mobile/',
    ),
    'event::en_GB' => array(
        'http://www.mysite.com/event/',
    ),
),
```

Warning: If your main context (the first) has a `fr/example.html` page and your `main::fr_FR` context has an `example.html` page, their URLs are identical (ie: `http://www.mysite.com/fr/example.html`). In this situation, only the page of your main context will be accessible.

Contexts on domains

```
<?php

'contexts' => array(
'main::en_GB' => array(
    'http://www.monsite.com/',
),
'main::fr_FR' => array(
    'http://www.mysite.fr/',
),
'main::es_ES' => array(
    'http://www.monsite.es/',
),
'mobile::fr_FR' => array(
    'http://mobile.monsite.fr/',
),
'event::en_GB' => array(
    'http://event.monsite.com/',
),
),
```

Note: Of course, your domains should also be properly configured in **Apache**.

Contexts with multiple URLs

```
<?php
'contexts' => array(
        'main::en_GB' => array(
                'http://www.monsite.com/',
        ),
        'main::fr_FR' => array(
                'http://www.mysite.fr/',
        ),
        'main::es_ES' => array(
                'http://www.monsite.es/',
        ),
        'mobile::fr_FR' => array(
                'http://mobile.monsite.fr/',
                'http://www.monsite-mobile.fr/',
                'http://www.mysite.com/mobile/',
        ),
        'event::en_GB' => array(
                'http://event.monsite.en/',
        ),
),
```

Going live

You will probably need to define, for each of your contexts, different URLs between your local (development) and production instances. You can do that using *environment-specific config files*.

Novius OS

A sample configuration file is available in `local/config/config.php.sample`. Just rename (or copy) it to `local/config/config.php`, and update it to your case.

cache

`Boolean` for use of cache on front. By default is `true`, except in DEVELOPMENT environment. New in version 0.2.0.2.

`cache_duration_page`
`Int`, number of seconds of cache validity. By default is `60`.

`cache_model_properties`
`Boolean` for use of cache on `Nos\Orm\Model` properties. By default is `false`. If is `true`, all models properties will be cached with a auto-refresh mechanism : if you add a column on a model which has properties defined, the cache will be refreshed by a DB `list_columns` request when you access this column with `get()` or `set()`. New in version Chiba: 1.0.1

`upload`
`Array`:

Parameters

- **disabled_extensions** – `Array` of invalid extensions for files uploaded in Novius OS. By default `php` is disabled.

`assets_minified`
`Boolean` for use of assets minified in back-office. By default is `true`, except in DEVELOPMENT environment.

Database

The DB configuration was initially created by the installation wizard. But you can (and you must to go live) update it after installation.

The DB configuration is in `local/config/db.php`.

See Also:

FuelPHP Database documentation for details.

Email

By default, your Novius OS is not configured to send mail, because it's too dependent on the server.

A sample configuration file is available in `local/config/email.config.php.sample`. Just rename (or copy) it to `local/config/email.config.php`, and update it to your case.

See Also:

FuelPHP email package documentation for details.

1.1.2 Application configuration

PHP to Javascript

Application configuration is mostly done in PHP, but a lot of code also runs in the browser, and is powered by JavaScript.

Some PHP configuration patterns have been created to define JavaScript behaviours from a PHP configuration file.

PHP nosActions

`$container.nosAction()` executes an action bound to a DOM element.

To define an action in PHP:

```php
<?php
'action' => array(
        'action' => 'actionName', // nosTabs, nosDialog, confirmationDialog, nosAjax, window.open, do
        // Other keys are array and depends on the actionName
),

// Example with nostabs
'action' => array(
        'action' => 'nosTabs',
        'tab' => array(
                'url' => 'admin/nos/page/page/insert_update/{{id}}',
                'label' => '{{title}}',
        ),
),
```

This syntax is used to define actions for:

- *Launchers*
- *Data catchers*
- buttons and menus in the *Appdesk*
- *Crud Controller*

PHP cellFormatters

Used to format the value displayed in a column of a grid in the *Appdesk*.

A `cellFormatter` is an associative array.
You can have multiple `cellFormatter`, just use arrays.
You can assign a key to a `cellFormatter`. This way, someone else can delete or modify it by overloading configuration.

Common keys

type The `cellFormatter` type. Can one of `bold`, `css`, `icon`, `iconClasses` or `link`.

replace If `true`, `cellFormatter` empties the current content.

Types

bold Formats the text in bold. No additional key.

css Apply CSS styles to the content.

css An associative array of all CSS styles to apply.

```
<?php
array(
        'type' => 'css',
        'css' => array(
                'text-align' => 'center',
        ),
),
```

icon Prepends an icon to the text, using an URL.

column Use a `data_mapping` column to fetch the icon URL.

src The icon URL.

mapping A mapping array to fetch URL depending on the value of the column.

size Force a size in pixels for the icon. Used for both width and height.

```
<?php
array(
        'type' => 'icon',
        'column' => 'column_icon', // URL is retrieved from the 'column_icon' column
        'size' => 16
),

// Or
array(
        'type' => 'icon',
        'src' => 'static/path/icon.png',
),

// Or
array(
        'type' => 'icon',
        'mapping' => array(
                '1' => 'static/path/icon-1.png', // If column value is '1', use this URL
                '2' => 'static/path/icon-2.png',
        ),
),
```

iconClasses Prepends an icon to the text, using CSS classes.

column Use a `data_mapping` column to fetch the icon CSS classes.

classes The icon CSS classes.

```
<?php
array(
        'type' => 'iconClasses',
        'column' => 'column_icon_classes', // CSS classes are fetch from the 'column_icon_classes' co
),

// Or
array(
        'type' => 'iconClasses',
        'classes' => 'icon icon-foo',
),
```

link Wraps a link to the text (which performs an action upon click).

action Action to perform when the link is clicked. Can be `default` to use the default action of the item, an *action name* of the item or a *nosAction*.

```
<?php
array(
        'type' => 'link',
        'action' => 'default', // Binds the default action (e.g.: 'edit the item' in the most of the
),

// Or
array(
        'type' => 'link',
        'action' => 'Namespace\Model_Example.result', // Binds the 'result' action of the item, whic
),

// Or
array(
        'type' => 'link',
        'action' => array(
                'action' => 'nosTabs', // Open a new tab
                'tab' => array(
                        'url' => 'admin/nos/page/page/insert_update/{{_id}}', // {{_id}} will be rep
                        'label' => '{{_title}}', // {{_title}} will be replaced by the item's title
                ),
        ),
),
```

Full example

```
<?php
'data_mapping' => array(
        'column_a' => array(
                'title' => 'Column a'
                'cellFormatters' => array(
                        'bold' => array(
                                'type' => 'bold',
                        ),
                        'center' => array(
                                'type' => 'css',
                                'css' => array(
                                        'text-align' => 'center',
                                ),
                        ),
                ),
        ),
        'column_b' => array(
                'title' => 'Column b'
                'cellFormatters' => array(
                        'icon' => array(
                                'type' => 'icon',
                                'column' => 'column_icon',
                                'size' => 16
                        ),
                ),
        ),
        'column_icon' => array(
                value => function($item) {
                        return $item->icon();
                },
        ),
),
```

Metadata

The `metadata.config.php` file is how an application is defined. It tells what contains the application and what it does.

The most important keys are:

name The name of the application.

namespace In which PHP namespace all the classes of the application must be defined.

icons In the 3 standard sizes 16*16, 32*32 and 64*64.

requires Optional. Which applications does your application requires. Array or string (in the last case, considered as an array with a unique element).

```php
<?php

return array(
    'name'      => 'Webpages',
    'namespace' => 'Nos\Page',
    'version'   => '0.2',
    'provider'  => array(
        'name'  => 'Novius OS',
    ),
    'extends' => array(
        // Optional,
    ),
    'requires' => array(
        // Optional
    ),
    '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',
    ),
    'permission' => array(
        // Empty array for now. Leave it.
    ),
    'i18n_file' => 'noviusos_page::metadata',
    'launchers' => array(
        // Optional
    ),
    'enhancers' => array(
        // Optional
    ),
    'templates' => array(
        // Optional
    ),
    'data_catchers' => array(
        // Optional
    ),
);
```

An application provides:

Extends

Application can extend other applications. The scope of the extension depends on the configuration you provide. The value of the *extends* key can be an associative array:

application Application that is extended

extend_configuration Optional, default value is true. Boolean defining if configuration files are extended (the extended application's configuration files are recursively merged with those of the extending application).

Launchers

A *launcher* is an icon on the home tab.

A launcher is defined by an associative array. Key is launcher ID, launcher properties is an associative array:

name Text to display for the icon.

icon Optional. URL to a 64*64 image, default will use the 64*64 icon of the app.

action What is done when clicking on the launcher. See *PHP nosActions*.

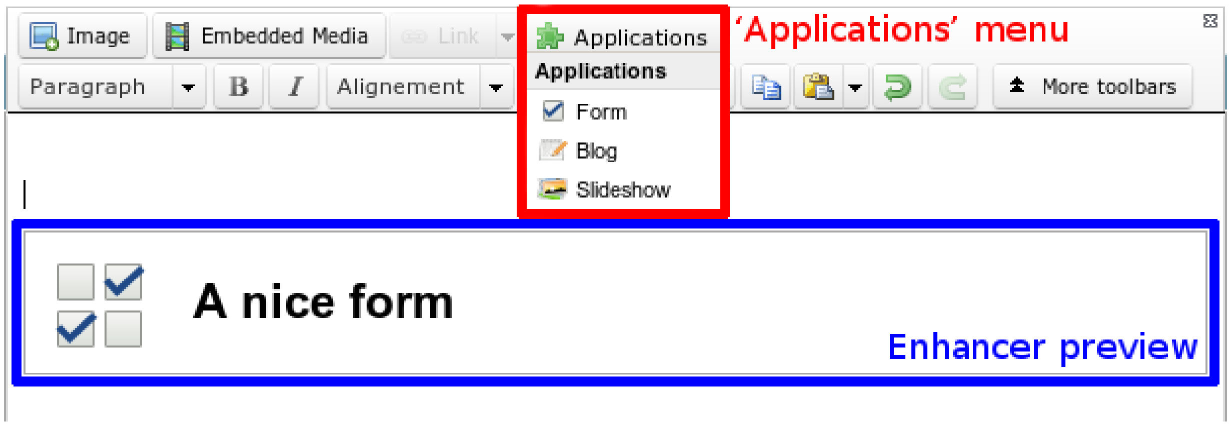
```php
<?php

return array(
    'launchers' => array(
        'noviusos_page' => array(
            'name' => 'Webpages',
            // 'icon' is not set, so the default icon will be used
            'action' => array(
                // open a tab
                'action' => 'nosTabs',
                'tab' => array(
                    'url' => 'admin/noviusos_page/appdesk/index',
                    // 'iconUrl' is not set, so the default icon will be used
                ),
            ),
        ),
    ),
);
```

Enhancers

Enhancers are used in WYSIWYG editors. They provide functionalities for the front-office.

For example, the 'Forms' application allows users to insert forms in their web pages (using an enhancer).

URL enhancers, a specific type of enhancers, handle their own URLs. For example, every blog post has an URL.



An enhancer is defined with:

title Title of the enhancer displayed when opening the 'Application' menu from the wysiwyg.

desc Optional. Description of the enhancer.

iconUrl Optional. URL to a 16*16 icon, displayed when opening the 'Application' menu from the wysiwyg, default will use the 16*16 icon of the app ;

enhancer URL of the front-office controller used to render the enhancer.

urlEnhancer Same that `enhancer`.Only one of the two keys can is used, depending if you want an URL enhancer or just a plain regular enhancer.

previewUrl Optional. URL of the controller used to render the preview in the wysiwyg.

dialog Optional. If you want a configuration popup, URL of the controller used to display and save the enhancer configuration. See `$container.nosDialog()` for the list of parameters.

```php
<?php

return array(
    'noviusos_form' => array(
        'title' => 'Form',
        'desc'  => '',
        // Here it's just a regular enhancer
        'enhancer' => 'noviusos_form/front/main',
        //'urlEnhancer' => 'noviusos_form/front/main',
        'iconUrl' => 'static/apps/noviusos_form/img/icons/form-16.png',
        // We'll use our controller to generate the preview
        'previewUrl' => 'admin/noviusos_form/enhancer/preview',
        // And the user has to configure it
        'dialog' => array(
            'contentUrl' => 'admin/noviusos_form/enhancer/popup',
            'width' => 450,
            'height' => 400,
            'ajax' => true,
        ),
    ),
);
```

Templates

Templates are similar to other CMS' templates or themes. They provide a layout for the front-office.

In Novius OS, a template contains one or more WYSIWYG editable area(s), which are is placed inside a grid.

The grid has a size of `cols * rows`, and each editable area is positioned using absolute coordinates (it's similar to `position: absolute` in CSS).

Each WYSIWYG editable area has:

- a **name**: it's the key in the `layout` array (see below) ;
- a **position**: absolute coordinates inside the grid (similar to `left` and `top` in CSS) ;
- a **size**: similar to `width` and `height` in CSS.

In the end, a template is defined with:

file path to the template file (it's a view)

title title of the template, it's shown when selecting a template for a page

cols grid width (in units)

rows grid height (in units)

layout list of the WYSIWYG editors inside the grid :

- the key is the name of the WYSIWYG ;
- the value is a comma-separated string containing (in this order) :
  - the left position (0-indexed) ;
  - the top position (0-indexed) ;
  - the width (in units) ;
  - the height (in units).

Here's an example:

```php
<?php

return array(
    'templates' => array(
        'top_menu' => array(
            'file' => 'noviusos_templates_basic::top_menu',
            'title' => 'Default template with a top menu',
            'cols' => 1,
            'rows' => 1,
            'layout' => array(
                // There is one WYSIWYG named 'content'
                // Position inside the grid: <left>,<top>,<width>,<height>
                'content' => '0,0,1,1',
            ),
        ),
    ),
);
```

Data catchers

Common

Configuration for `Nos\Orm\Model`, used in *Appdesk*, *Crud Controller* or *Inspectors*.

Associative array:

data_mapping columns on *Appdesk* and *Inspectors*.

i18n Optional, common translation

actions Optional, common actions on the `Nos\Orm\Model`.

icons Optional, common icons related to the `Nos\Orm\Model`.

Data mapping

Associative array where each key => value defines a column, all keys are optionals.

title Title of the grid column. If not set, column will not be displayed.

column Default value is same as key.

search_column Default value is column key value. Defines on which SQL column search / order.

search_relation Default value is deduced from key (ex: `rel->col`). Relation to load (via related function on query).

sorting_callback A closure function taking two parameters: the `$query` object and the `$sortDirection`.

multiContextHide Hide grid column when items are filtered on more than one contexts.

value A closure function taking current item `Nos\Orm\Model` in first parameter. Overloads value displayed in the grid.

cellFormatters Associative array of *cellFormatters* for formatting column display.

```php
<?php
return array(
    'data_mapping' => array(
        'column_a' => array(
            'title' => 'Column A',
            'column' => 'col_a',
            'multiContextHide' => true,
            'cellFormatters' => array(
                'center' => array(
                    'type' => 'css',
                    'css' => array('text-align' => 'center'),
                ),
            ),
        ),
        'column_b' => array(
            'title' => 'B',
            'search_column' => 'col_b_search',
            'search_relation' => 'rel',
            'value' => function($item) {
                return 'test';
            },
        ),
        // ...
    ),
);
```

Particular cases In the following example, `column_a` is sent in json but is not displayed in the grid.

```
<?php
return array(
    'data_mapping' => array(
        'column_a',
    ),
);
```

In the following example, `col_b` is sent in json under the `column_b` key but is not displayed in the grid.

```
<?php
return array(
    'data_mapping' => array(
        'column_b' => 'col_b',
    ),
);
```

If the `Model` has the `Orm_Behaviour_Twinnable` behaviour, a pseudo column `context` is automatically added at the end of the `data_mapping`. But, if you want to place it elsewhere, you can force its position like this:

```
<?php
return array(
    'data_mapping' => array(
        'column_a' => array(
            'title' => 'Column a'
        ),
        'context',
        'column_b' => array(
            'title' => 'Column b'
        ),
    ),
    // ...
);
```

I18n

This key contains all the common translations.

```
<?php
return array(
    'i18n' => array(
        // Crud
        'notification item added' => __('Done! The item has been added.'),
        'notification item saved' => __('OK, all changes are saved.'),
        'notification item deleted' => __('The item has been deleted.'),

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

        // ... see the 'framework/config/i18n_common.config.php' file to include the appropriate key
    ),
);
```

Actions

This key contains all the common actions related to the model. 5 actions which are automatically added:

add The *Add model* button located on the appdesk's toolbar

edit The *Edit* button located on the grids and the crud's toolbar

delete The *Delete* button located on the grids and the crud's toolbar

visualise The *Visualise* button located on the grids and crud's toolbar, if the item can be displayed in front-office.

share The *Share* button located on the crud's toolbar, if the item has the `Nos\Orm_Behaviour_Sharable` behaviour.

The action key can be filled in two different ways.

The most common way is to use an associative array:

```php
<?php
return array(
    // ...
    'actions' => array(
        'action_1' => array(/* configuration */),
        'action_2' => array(/* configuration */),
        // ...
    ),
);
```

If you want to change the order in which the actions are displayed, two keys are to be defined:

list associative array of actions (similar to the previous `actions` key)

order array of action's key defining their order

```php
<?php
return array(
    // ...
    'actions' => array(
        'list' => array(
            'action_1' => array(/* configuration */),
            'action_2' => array(/* configuration */),
            // ...
        ),
        'order' => array(
            'action_2',
            'action_1'
        ),
    ),
);
```

Each action is an associative array. Key is the action ID, and value is an array defining the action configuration:

action which action is executed when clicking on the button (using *nosAction*).

label Text associated with the action (either shown as text or in a tooltip).

primary Is it a primary action? Primary actions have a dedicated button, and secondary actions appears in the action drop down.

icon Icon of the action. It's a jquery ui icon class, but without the leading `ui-icon-` string.

red Is it a red action? (especially used for 'Delete')

targets Where is the action displayed? This has no effect if the action is not `visible` (see below). There are 3 locations available:

grid Is the action displayed on the grid (appdesk and inspector)?

toolbar-grid Is the action displayed on the grid's toolbar?

toolbar-edit Is the action displayed on the crud's toolbar (edition form)?

disabled Callback function or array of callback functions (the one taken into account is the first which doesn't return `false`) that returns a boolean or a string defining if the action is disabled or not for an item (a string disable the action and the value is used as title). There is two sent parameters: the current `$item`, and `$params` containing the common configuration.

visible Callback function or array of callback functions (the one taken into account is the first which doesn't return `true`) that returns a boolean defining if the action is visible or not on a context. There is only one parameter sent, an associative array:

model Model tested.

target Target where action is displayed. Value can be `grid`, `toolbar-grid` or `toolbar-edit` (related to action's `targets`).

item Only populated when `target == 'toolbar-edit'`.

class Name of the controller class calling the function (this way you can differentiate appdesk and inspectors for instance).

```php
<?php
return array(
    'actions '=> array(
        'action_id' => array(
            'action' => array(
                'action' => 'confirmationDialog',
                'dialog' => array(
                    'contentUrl' => '{{controller_base_url}}delete/{{_id}}',
                    'title' => 'Delete',
                ),
            ),
            'label' => __('Delete'),
            'primary' => true,
            'icon' => 'trash',
            'red' => true,
            'targets' => array(
                'grid' => true,
                'toolbar-edit' => true,
            ),
            'disabled' => function($item) {
                return false;
            },
            'visible' => function($params) {
                return !isset($params['item']) || !$params['item']->is_new();
            },
        ),
    ),
);
```

Default actions and particular cases Default actions (such as `add` or `edit`) can be overloaded with this `actions` key. Arr::merge is used to merge defined actions with default actions.

To hide an action, set its value to `false`:

```php
<?php
return array(
    // ...
    'actions '=> array(
        'add' => false,
    ),
);
```

Placeholders Placeholders are available in order to simplify action targets and labels. First, some placeholders are always available:

- `controller_base_url`: URL of the crud controller
- `model_label`: generated from model class name
- `_id`: ID of the item
- `_title`: Title of the item
- `_context`: if the item has the `Contextable` behaviour

Additionally, all `dataset` keys can be used as placeholders.

Icons

This key contains all common icons related to the model. Structure is similar to the icons key of the *Metadata* configuration file :

```
<?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',
    ),
);
```

Appdesk

Configuration for `Nos\Orm\Model`'s appdesk.

Associative array:

model Model name.

query Optional. Additional informations about the query.

search_text Optional. Array of columns in which we search when the user fills appdesk's search bar.

data_mapping Optional. Defines which data_mapping items are displayed.

inspectors Optional.

views Optional.

selectedView Optional. Default selected view identifier.

inputs Optional. How to process additional parameters sent to the appdesk.

i18n Optional. Extends default text items.

thumbnails Optional, boolean. Can the appdesk display items as thumbnails ?

tree Optional (automatically filled when model has the *Tree* behaviour enabled).

appdesk Optional. Additional display information about the appdesk.

query

Associative array. All keys are optional. Most keys are similar than the find function second parameters.

model Model on which query is executed.

limit

order

related

callback Array of callback functions allowing you to customize the query (first parameter is the current query, must return the modified query).

data_mapping

Associative or simple array. Defines which data_mapping items from *Common* configuration we display (mostly filtering).

Is is also possible to define new custom data_mapping items which will be only used on the appdesk.

If only value is defined, appdesk will display the data_mapping item from common configuration.

If it is a key => value association (value must then be an array), then the data_mapping item from common configuration is extended by the value.

```
<?php
return array(
    // ...
    'data_mapping '=> array(
        'col1', // will display the col1 data_mapping item from common configuration.
        'col2' => array( // if col2 exists on common configuration data_mapping key then it is extend
            // ...
        ),
    ),
);
```

inspectors

Associative or simple array. Defines which inspectors are used on appdesk.

If is also possible to define new custom inspectors which will be only used on the appdesk.

If only value is defined: * appdesk will display the inspector located at *inspector/inspector_name* if the value is *inspector_name*. * it also supports inspector class names (*MyAppController_Admin_Inspector_Name*).

If it is a key => value association (value must then be an array), then the inspector configuration is extended by the value.

```
<?php
return array(
    // ...
    'inspectors '=> array(
        'inspector1', // will display the inspector located at inspector/inspector1.
        '\My\App\Controller_Admin_Inspector_Name', // will display the inspector which class is \My\A
        'inspector2' => array( // if inspector/inspector2 exists, then it is extended ; otherwise it
            // ...
        ),
    ),
);
```

views

Associative array defining different way of displaying the appdesk. The key is the view identifier. Value is view configuration:

name Optionnal. Display view name in view selector

virtual Optionnal. Is the view present on the view selector ?

json Array of javascript files to load. These javascript extends appdesk configuration.

inputs

How to process additional parameters sent to the appdesk. Associative array, to define a callback for each parameter.

```
<?php
return array(
    // ...
    'inputs' => array(
        'monk_species_id' => function($value, $query) {
            // ...
            return $query;
        },
    ),
);
```

thumbnails

Can the appdesk display items as thumbnails ?

If defined to true, data_mapping has to define two keys:

thumbnail url of item thumbnail.

thumbnailAlternate Default thumbnail when there is no thumbnails or thumbnail can't be found.

tree

Defines how the model tree is constructed on the appdesk. It is automatically filled when model has the *Tree* behaviour. Associative array:

models Models to be loaded on the tree. Array of associative array:

> **model** Model class name
>
> **order_by**
>
> **childs** Array of model class name. Which models instances are children.
>
> **dataset** dataset information sent by objects in json format.

callback Array of callback functions allowing you to customize the query (first parameter is the query).

roots

model Model class name

order_by

where

appdesk

Associative array describing how appdesk interacts and is displayed. All items are automatically generated, but can be overloaded.

appdesk Defines how appdesk is displayed. Associative array:

defaultView Default view of appdesk. It can be 'grid' (default), 'treeGrid' or 'thumbnails'.

buttons Associative array containing grid toolbar actions information. See *Actions*.

splitterVertical Size of the vertical splitter.

inspectors Associative array containing information about inspectors. Key is the inspector identifier, value is its configurations. See *Inspectors* configuration.

grid Grids informations. Associative array:

urlJson Url of the json API to get items

columns Columns informations

treeGrid

urlJson Url of the json API to get items

tab Tab information (see *nosTabs*).

reloadEvent Event name that will reload appdesk.

actions Associative array containing main grid actions information. See *Actions*.

Inspectors

Configuration for an inspector. Associative array depending on inspector type.

The only common keys are *input*, which describes how the inspector will affect the main grid, and *appdesk*, which defines how it is displayed.

input

Associative array:

key Key name on which the main grid will be filtered. By default, if query key is not defined, must be the column name on which you want to filter the model.

query Optional. Callback function defining how to interact with the main grid query. The function gets two parameters ; the first is the value sent by the inspectors (when items are selected), and the second is the current query. The function must return the transformed query.

appdesk

Associative array. All keys are optional:

- **label** Main label of the inspector.
- **contextChange** Does the inspector reloads when user selects an other context.
- **reloadEvent** Event name that will reload the inspector?
- **vertical** Is the appdesk vertical or horizontal ?
- **inputName** Key on which selected values are sent.

Model inspector

Associative array:

- **model** Model class name
- **query** Optional. Additional informations about the query. See *query in appdesk configuration*.
- **data_mapping** Optional. Defines which data_mapping items are displayed. See *data_mapping in appdesk configuration*.
- **appdesk** Optional. Additional display information about the appdesk.

appdesk Associative array:

- **grid** Optional. Grids informations. Associative array:
    - **urlJson** Url of the json API to get items
    - **columns** Columns informations.

Model tree inspector

Associative array:

- **model** Model class name
- **data_mapping** Optional. Defines which data_mapping items are displayed. See *data_mapping in appdesk configuration*.
- **appdesk** Optional. Additional display information about the appdesk.
- **models** Optional. See *appdesk tree configuration*.
- **roots** Optional. See *appdesk tree configuration*.
- **root_node** Optional. Add a global root node. Associative array containing dataset values (columns to be displayed).

appdesk Associative array:

- **treeGrid** Optional. Grids informations. Associative array:
    - **urlJson** Url of the json API to get items
    - **columns** Columns informations.

Date inspector

Associative array:

input_begin Name of the input for the begin date. Default value: `date_begin`.

input_end Name of the input for the end date. Default value: `date_end`.

labels

- **Custom dates** Default value: `Custom dates`.
- **from begin to end** Default value: `from {{begin}} to {{end}}`.
- **until end** Default value: `until {{end}}`.
- **since begin** Default value: `since {{begin}}`.

options

Array of options displayed by inspector. Default value: `array('custom', 'since', 'month', 'year')`.

since

- **optgroup** Label of this group in the inspector. Default value: `Since`.
- **options** Associative array. Key is the date, string is processed by Date Class, value is the label of the date.

month

- **optgroup** Label of this group in the inspector. Default value: `Previous months`.
- **first_month** Month to start list from. Default value: `now`.
- **limit_type** Limit type where the list end (value can be `year` or `month`). Default value: `year`.
- **limit_value** Number of items to display. For example, if `limit_type` is `month` and `limit_value` is 5, it will display the last 5 months. Default value: `1`.

year

- **optgroup** Label of this group in the inspector. Default value: `Years`.
- **first_year** Year to start list from. Default value: `now`.
- **limit** Number of years to display. Default value: `4`.

Plain data inspector

Displays static data. Associative array:

data Array of items. Each item is an associative array:

- **id**
- **title**
- **icon** Optional.

input

- **query** Here this key is mandatory.

appdesk

url Url to load in order to display list extension.

grid How is the inspector grid displayed

columns Grid columns. Associative array, key is column identifier and value is an associative array:

headerText Column title

visible Is key visible

dataKey For each data item, defines which key is displayed

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.

controller_url Url of the CRUD controller

model Which model it generates the form for.

environment_relation Relation name. Allows to define a children / parent relationship.

tab Tab informations (such as the icon & label), see *nosTabs*

views Optional. Views locations.

layout How the form looks like and where fields are displayed inside it. Optional if *layout_insert* and *layout_update* are defined.

layout_insert Optional. Specific layout for insert. Will use `layout` as default value.

layout_update Optional. Specific layour for update. Will use `layout` as default value.

fields All fields displayed in the form.

```php
<?php
return array(
        'controller_url' => '',
'environment_relation' => null,
        'model' => '',
'tab' => array(
    'iconUrl' => '',
    'labels' => array(
        'update' => null,
        'insert' => 'New item',
        'blankSlate' => 'Translate an item',
    ),
),
        'layout' => array(),
        'fields' => array(),
'require_js' => array(),
'views' => array(
    'form' => 'nos::crud/form',
    'delete' => 'nos::crud/delete_popup',
),
);
```

environment_relation

The `environment_relation` must contain a relation name of type `Orm\\BelongsTo`.

It allows to use the `GET['environment_id']` in an action of this model.

It will be used to populate the column associated with the relation. Examples include:

- Add a page at a specific location inside the tree (with a pre-selected parent) ;
- Add a media in a specific folder ;
- Add a monkey inside a species ;
- Add an blog post inside a category.

Some examples:

```php
<?php
// Add a sub-page action
'action' => array(
    'action' => 'nosTabs',
    'tab' => array(
        // The 'page_parent_id' will be populated with $_GET['environment_id'] from the action
        'url' => '{{controller_base_url}}insert_update?environment_id={{_id}}&context={{_context}}',
    ),
),


// Add a media in this folder action
'action' => array(
    'action' => 'nosTabs',
    'tab' => array(
        // The 'media_folder_id' will be populated with $_GET['environment_id'] from the action
        'url' => '{{controller_base_url}}insert_update?environment_id={{id}}',
    ),
),
```

tab

See Also:

nosTabs

The `tab` configuration array has a special `labels` key, to handle several `label` depending on the case.

insert Adding an new item

blankSlate Translating an existing item

update Editing an existing item

- `insert` and `update` must contain plain `string` value ;
- `update` can either contain a plain `string` value, or a `callable` taking one argument: the `$item` ;
- The default value for `labels.update` is the item's title.

```php
<?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 'blankSlate'
                'labels' => array(
                        'insert' => __('Add a monkey'),
                        'blankSlate' => __('Translate a monkey'),
                ),
        ),
);
```

views

form View for the form (both insert and update). Default is `nos::crud/form`.

delete View for the delete popup. Default is `nos::crud/delete_popup`.

insert Optional. View for the insert form (will use `form` value as default)

update Optional. View for the update form (will use `form` value as default)

layout

The `layout` is a data passed to the parameters of the view. It list all the views needed to render the form.

There are two syntaxes:

- the full syntax ;
- a simplified syntax, which is used 90% of the time.

The **full syntax** for using a layout is the following:

```php
<?php
'layout' => array(
    'first_view' => array(
        'view' => 'nos::form/layout_standard',
        'params' => array(
            // View data (depends on the view).
            'title' => '',
            'content' => '',
        ),
    ),
    'second_view' => array(
        'view' => 'noviusos_page::admin/page_form',
        // No 'params'
    ),
    // More views can be used here.
),
```

In addition to view-specific params / data, 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.

Because 90% of the time, we want to use `nos::form/layout_standard` as the view for the layout, a **simplified syntax** was created: only write the view `params` of the standard layout.

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 90% of the time.

```php
<?php
'layout' => array(
    // View data
    'title' => '',
    'content' => '',
),
```

We only need to define the view data for the standard layout, and it will be wrapped like so:

```php
<?php
$layout = array(
    array(
        'view' => 'nos::form/layout_standard',
        'params' =>  $layout,
    ),
);
```

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

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

Native views included in Novius OS

- Used as **container** for other layouts / views
  - *Standard layout*: used as a container for other views ;
  - *Expander*: used inside `layout_standard.content` in the Pages application ;
- Used as **final** views:
  - *Fields*: used inside `layout_standard.content` in the User application ;
  - *Accordion*: used inside `layout_standard.menu` in the Pages application.

See Also:

Views

fields

Contains the fields definition array.

The `fields` syntax is based on an existing FuelPHP feature, which is used to configure form attributes for each column of a Model :

See Also:

FuelPHP documentation on Model::$_properties

In addition to standard form fields, Novius OS has *renderers*, which are a bit more advanced. For instance, they allow to select a media, a page, a date...

The field name is determined using the key. Then, for each field:

label Text for the label. Won't be shown for hidden fields

form `array` Attributes of the <input> tag

renderer Class name of a renderer

renderer_options (optional) `array` Options for the renderer

validation (optional) `array` rules used to validate the field.

expert (optional) `boolean` Should the field be visible only to expert users? Default `false`.

show_when (optional) `callable` Custom callback function to alter the visibility of the field. Must return `true` for the field to be shown.

To choose how the field is displayed, you only need to specify either `form` (a native HTML `<input>`) or a `renderer` (like a date picker or a wysiwyg), you don't need both. If both keys are filled, the renderer will be used to display the field (and the `form` key will be ignored).

Configuration example:

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

Standard fields Bold text is the value for the `type` property.

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

The `tag` property can be used to force a precise HTML tag, it's useful for a `submit` button.

FuelPHP will automatically use the `value` as the button's text.

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

Renderers (advanced fields) The renderer list is available *in a dedicated page*.

Glossary

Launcher Icon on the *home tab*. It's often the most visible part of an application.

Enhancer They are used in WYSIWYG editors abd provide functionalities for the front-office.

URL Enhancer A specific type of enhancers, which handle its own URLs. For example, every blog post has dedicated URL.

Template They are similar to other CMS' templates or themes and provide a view for the front-office.

Permissions

This configuration file is used to define the permissions used in your application.

It has the same syntax as the **full** version of the *crud layout*.

i.e. You need to configure one or several (`view` + `params`) pair(s), like so:

```
<?php
return array(
    'all' => array(
        'view' => 'nos::form/accordion',
        'params' => array(
            'accordions' => array(
                'standalone' => array(
                    'title' => __('Permissions for this application'),
                    'view' => 'nos::admin/permissions/standalone',
                    'params' => array(
                        'list' => array(
                            'noviusos_app::create' => array(
                                'label' => __('Can create new items'),
                                'checked' => true,
                            ),
                            'noviusos_app::delete_locked' => array(
                                'label' => __('Can delete locked item'),
                            ),
                        ),
                    ),
                ),
            ),
        ),
    ),
);
```

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

- `$role` : Current `role` for which the permissions are applicable to.

We (kindly) provide a native view to cover basic usage: *Permissions standalone*.

1.1.3 Constants

All native constants of FuelPHP are available but links to specific Novius OS directories.

See Also:

FuelPHP native constants documentation.

constant NOSROOT
Path where Novius OS is installed.

constant NOSPATH
Path of Novius OS core: `NOSROOT/novius-os/framework/`.

constant APPPATH
Overload of the native FuelPHP's constant. Path to the application directory: `NOSROOT/local/`.

constant COREPATH
Overload of the native FuelPHP's constant. Path to the FuelPHP core directory: `NOSROOT/novius-os/fuel-core/`.

constant DOCROOT
Overload of the native FuelPHP's constant. Path to the location where the *startup script* is: `NOSROOT/public/`.

constant PKGPATH
Overload of the native FuelPHP's constant. Path to the packages directory: `NOSROOT/novius-os/packages/`.

1.1.4 Models

Model

class `Nos\Orm\`**`Model`**
Extends *Model of FuelPHP ORM*.

Novius OS Model have some differences compare with FuelPHP Model :

- Novius OS implements a cache mechanism for properties when they are fetched from the database. By default, cache files are save in `NOSPATH/local/cache/fuelphp/list_columns`.
- In property definition, put `convert_empty_to_null` key to `true` if you want that this property stores a null value when it receives empty string.

Configuration

property `Nos\Orm\Model::$`**`title_property`**

Defines the title property of a model. Can be a column name or a closure (which take current `$iitem` as parameter).

If not defined, automatically set to first column which has `title`, `label` or `name` in its name, or (as last resort) the first `varchar`.

property `Nos\Orm\Model::$`**`behaviours`**
Defines the behaviours of model. Same syntax as *observers*.

property `Nos\Orm\Model::$`**`attachment`**
Defines the attachments of a model. Attachment is a special type of *relations* created for Novius OS. See `Nos\Attachment`.

In Novius OS, you can configurate model by a file configuration. For sample: if in your application you define a `Model_Monkey` class, you can create a file `config/model/monkey.config.php` to extend configuration. All this attributes can be defined in configuration file : `properties`, `table_name`, `title_property`, `observers`, `behaviours` and all relations types (`has_many`, `belongs_to`, `has_one`, `many_many` and `attachment`).

Examples Example in the class definition:

```php
<?php
class Model_Example extends \Nos\Orm\Model
{
        // In this example, attachments use defaults properties
        protected static $_attachment = array(
                'avatar' => array(),
                'cv' => array(),
        );
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Contextable' => array(
                        'events' => array('before_insert'),
                        'context_property' => 'ex_context',
                ),
        );

        // In this example, use a column name for defined title_property
        protected static $_title_property = 'ex_reference';
```

Example in configuration file:

```
<?php
return array(
        'attachment' => array(
                'avatar' => array(
                        'dir' => 'namespace/model_name/avatar/',
                        'image' => true,
                        'alias' => 'avatar',
                ),
                'curriculum_vitae' => array(
                        'dir' => 'namespace/model_name/curriculum_vitae/',
                        'alias' => 'cv',
                        'extensions' => array('doc', 'odt', 'pdf'),
                        'check' => array('ClassName', 'methodName'),
                ),
        ),

        'behaviours' => array(
                'Nos\Orm_Behaviour_Contextable' => array(
                        'events' => array('before_insert'),
                        'context_property' => 'ex_context',
                ),
        ),

        // In this example, use a closure for defined title_property
        'title_property' => function($item) {
                return $item->ex_reference;
        },
);
```

Relations

property `Nos\Orm\Model::$`**`linked_wysiwygs`**

- Relation type: *has_many*.
- Model: `Nos\Model_Wysiwyg`

property `Nos\Orm\Model::$`**`linked_medias`**

- Relation type: *has_many*.
- Model: `Nos\Media\Model_Link`

Warning: Don't use these relations directly, we created accessors for them.

Accessors

property Nos\Orm\Model::$**medias**

Accessor for Nos\Media\Model_Link linked to model.

```
<?php
$item->medias->avatar; // Get a Model_Link named 'avatar'
$item->medias->avatar->media; // Get Model_Media named 'avatar'

$item->medias->cv = $Model_Media; // Attach a Model_Media named 'cv'

$item->medias->cv = null; // Detach a media from an item
// or
unset($item->medias->cv);
```

property Nos\Orm\Model::$**wysiwygs**

Accessor for Nos\Model_Wysiwyg linked to model.

```
<?php
$item->wysiwygs->content; // Get a Model_Wysiwyg named 'content'
$item->wysiwygs->content->wysiwyg_text; // Get content of Model_Wysiwyg named 'content'

$item->wysiwygs->summary = 'foo'; // Set a Model_Wysiwyg named 'content', with content 'foo'.

$item->medias->summary = null; // Remove a wysiwyg from an item
// or
unset($item->wysiwygs->summary);
```

Methods

static Nos\Orm\Model::**title_property**

Returns Title property of model. See Model::$title_property.

static Nos\Orm\Model::**behaviours**(*$specific = null*, *$default = null*)

Nos\Orm\Model::**get_possible_context**()

Returns Array of possible contexts ID for current item. See *Multi-Contexts*.

static Nos\Orm\Model::**add_properties**(*$properties*)

Parameters

- **$properties** (*array*) – Additional properties (merged).

static Nos\Orm\Model::**prefix**

Returns Prefix of column name. Computed from the primary key name (everything before the first _ character).

Nos\Orm\Model::**title_item**()

Returns Returns the item's title, calculated from Model::$title_property.

Nos\Orm\Model::**pick**(*$column*[, *$column*[, *$column*[, *...*]]])

Parameters

- **$column** (*array*) – A column name.

Returns Returns the first non empty column. Will add column prefix (see Model::prefix) when needed.

Model_Page

class `Nos\Page\`**`Model_Page`**

Extends `Nos\Orm\Model`.

Constants

Page types

constant `Nos\Page\Model_Page::`**`TYPE_CLASSIC`**

constant `Nos\Page\Model_Page::`**`TYPE_EXTERNAL_LINK`**

Possible values of the `page_type` column.

External target types

constant `Nos\Page\Model_Page::`**`EXTERNAL_TARGET_NEW`**

constant `Nos\Page\Model_Page::`**`EXTERNAL_TARGET_SAME`**

Possible values of the `page_external_link_type` column.

Lock types

constant `Nos\Page\Model_Page::`**`LOCK_UNLOCKED`**

constant `Nos\Page\Model_Page::`**`LOCK_DELETION`**

Possible values of the `page_lock` column.

Relations

property `Nos\Page\Model_Page::$`**`children`**

- Relation type: *has_many*.
- Model: `Model_Page`

property `Nos\Page\Model_Page::$`**`parent`**

- Relation type: *belongs_to*.
- Model: `Model_Page`

Behaviours

- `Twinnable`
- `Tree`
- `Virtual path`
- `Sortable`
- `Publishable`

Methods

static `Nos\Page\Model_Page::`**`link`**

Returns `string` The href and target attributes for an HTML link, such as `href="..." target="..."`.

static `Nos\Page\Model_Page::`**`url`**`(`*$params = array()*`)`

Parameters

- **$params** (*array*) –

  preview If set, returns a preview URL

Returns The absolute URL of the page

Media Models

Model_Media

class `Nos\Media\`**`Model_Media`**

Extends `Nos\Orm\Model`.

Relations

property `Nos\Media\Model_Media::$`**`folder`**

•Relation type: *belongs_to*.

•Model: `Model_Folder`

Behaviours

- `Virtual path`

Methods

`Nos\Media\Model_Media::`**`delete_from_disk`**`()`

Returns `True` or `false` depending on whether the deletion was successful.

Delete the original media file from the disk.

`Nos\Media\Model_Media::`**`delete_public_cache`**`()`

Returns `True` or `false` depending on whether the deletion was successful.

Delete all the cached versions (thumbnails) of the media files from the disk.

`Nos\Media\Model_Media::`**`get_path`**`()`

Returns Relative media file virtual path.

`Nos\Media\Model_Media::`**`get_public_path`**`()`

Returns Public media file URL relative to base href.

`Nos\Media\Model_Media::`**`get_private_path`**`()`

Returns Private media file path relative to Novius OS root directory.

`Nos\Media\Model_Media::`**`get_img_tag`**`(`*$params = array()*`)`

Parameters

- **$params** (*array*) –

  max_width Max width of the image.

  max_height Max height of the image.

Returns If the media file is an image, a HTML `<img>` tag with `src`, `width` and `height` attributes, depends of `$params`. `False` otherwise.

`Nos\Media\Model_Media::`**`get_img_tag_resized`**`(`*$max_width = null*, *$max_height = null*`)`

Parameters

- **$max_width** (*array*) – Max width of the image.

- **$max_height** (*array*) – Max height of the image.

Returns If the media file is an image, a HTML `<img>` tag with `src`, `width` and `height` attributes. `False` otherwise.

Alias of `get_img_tag(array('width' => $max_width, 'height' => $max_height))`.

`Nos\Media\Model_Media::`**`get_img_infos`**`(`*$max_width = null*, *$max_height = null*`)`

Parameters

- **$max_width** (*array*) – Max width of the image.
- **$max_height** (*array*) – Max height of the image.

Returns If the media file is an image, an associative array with keys `src`, `width` and `height` depends of size parameters. `False` otherwise.

`Nos\Media\Model_Media::`**`get_public_path_resized`**`(`*$max_width = 0*, *$max_height = 0*`)`

Parameters

- **$max_width** (*array*) – Max width of the image.
- **$max_height** (*array*) – Max height of the image.

Returns If the media file is an image, media URL relative to base href for specify size parameters. `False` otherwise.

`Nos\Media\Model_Media::`**`is_image`**`()`

Returns `True` or `false`, depend if media is an image.

Model_Link

class `Nos\Media\`**`Model_Link`**

Link between a `Model_Media` and a `Nos\Orm\Model`.

Extends `Nos\Orm\Model`.

Columns

medil_id Model_Link primary key.

medil_from_table DB table name of the model which the media is linked to.

medil_foreign_id ID of the model which the media is linked to.

medil_key `string` key identifying the linked media.

medil_media_id ID of the linked media.

Relations

property `Nos\Media\Model_Link::$`**`media`**

•Relation type: *belongs_to*.

•Model: `Model_Media`

Model_Folder

class `Nos\Media\`**`Model_Folder`**

Extends `Nos\Orm\Model`.

Relations

property `Nos\Media\Model_Folder::$`**`children`**

- Relation type: *has_many*.
- Model: `Model_Folder`

property `Nos\Media\Model_Folder::$`**`media`**

- Relation type: *has_many*.
- Model: `Model_Media`

property `Nos\Media\Model_Folder::$`**`parent`**

- Relation type: *belongs_to*.
- Model: `Model_Folder`

Behaviours

- `Tree`
- `Virtual path`

Methods

`Nos\Media\Model_Folder::`**`delete_from_disk`**`()`

Returns `True` or `false` depending on whether the deletion was successful.

Delete the folder and all its content (recursively).

`Nos\Media\Model_Folder::`**`delete_public_cache`**`()`

Returns `True` or `false` depending on whether the deletion was successful.

Delete all the public and cached entries (image thumbnails) of this folder

`Nos\Media\Model_Folder::`**`path`**`(`*$file* = ‘’`)`

Parameters

- **$file** (*string*) – A file name to append to the path.

Returns Absolute folder path.

`Nos\Media\Model_Folder::`**`count_media`**`()`

Returns Number of media contained in the folder and all its sub-folders.

`Nos\Media\Model_Folder::`**`count_media_usage`**`()`

Returns Number of media in use (by the applications) contained in this folder and all its sub-folders.

User Models

Model_User

class `Nos\User\`**`Model_User`**

Extend `Nos\Orm\Model`.

Relations

property `Nos\User\Model_User::$`**`roles`**

- Relation type : *many_many*.
- Model : `Model_Role`

`Nos\User\Model_User::`**`check_permission`**`(`*$permission_name*, *$category_key* = *null*`)`

Parameters

- **$permission_name** (*string*) – Name of the permission to check against

- **$category_key** (*string*) – (optional) If the permission has categories, the category key to check against

Returns `True` if the user has the required authorisation, `false` otherwise

Methods

`Nos\User\Model_User::`**`check_password`**`(`*$password*`)`

Parameters

- **$password** (*string*) – Password to check if it matches the user's password

Returns `True` if passwords match, `false` otherwise.

`Nos\User\Model_User::`**`generate_md5`**`()`

Generate a new `md5` for the user (store in the `user_md5` column).

Model_Role

class `Nos\User\`**`Model_Role`**

Extend `Nos\Orm\Model`.

Relations

property `Nos\User\Model_Role::$`**`users`**

- Relation type : *many_many*.
- Model : `Model_User`

Methods

`Nos\User\Model_Role::`**`check_permission`**`(`*$permission_name*, *$category_key = null*`)`

Parameters

- **$permission_name** (*string*) – Name of the permission to check against
- **$category_key** (*string*) – (optional) If the permission has categories, the category key to check against

Returns `True` if the role has the required authorisation, `false` otherwise

`Nos\User\Model_Role::`**`listPermissionCategories`**`(`*$permission_name*`)`

List all the categories of a given permission name. Returns an array of string or false when the role has not access, or the permission name does not exists.

param string $permission_name The name of the permission to retrieve categories from

param string $category_key (optional) If the permission has categories, the category key to check against

returns `array|false` An array containing the list of categories (values) for the request permission name

Model_Wysiwyg

class `Nos\`**`Model_Wysiwyg`**

Extend `Nos\Orm\Model`.

Columns

wysiwyg_id Model_Wysiwyg primary key.

wysiwyg_join_table DB table name of the model which the wysiwyg is linked to.

wysiwyg_foreign_id ID of the model which the wysiwyg is linked to.

wysiwyg_key `string` key identifying the linked wysiwyg.

wysiwyg_text Wysiwyg content.

Model_Content_Nuggets

class `Nos\`**`Model_Content_Nuggets`**
Extends `Nos\Orm\Model`.

Constants

constant `Nos\Model_Content_Nuggets::`**`DEFAULT_CATCHER`**
Default value for the `content_catcher` column.

Columns

content_id Model_Content_Nuggets primary key.

content_catcher `Data catcher` name.

content_model_name Name of the model this `content nuggets` originates from.

content_model_id ID of the model this `content nuggets` originates from.

content_data The `content nuggets`. Stored as a serialized associative array.

1.1.5 Behaviours

Contextable

class `Nos\`**`Orm_Behaviour_Contextable`**
Allows a `Nos\Orm\Model` to be bound to a context.

See Also:

Multi-Contexts.

Configuration

property `Nos\Orm_Behaviour_Contextable::$`**`context_property`**
Required. Column used to store the item's context. Its data type must be `varchar(25)`.

property `Nos\Orm_Behaviour_Contextable::$`**`default_context`**
Default context to use if not set when creating an item.

Methods

`Nos\Orm_Behaviour_Contextable::`**`get_context`**`()`

Returns The item's context.

Other

This behaviour extends *Model->find()*.

Add option to `where` array passed to method: you can use the `context` key as an alias to search in the column `Orm_Behaviour_Contextable::$context_property`.

Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Contextable' => array(
                        'events' => array('before_insert'),
                        'context_property' => 'form_context',
                ),
        );
}
```

Publishable

class `Nos\`**`Orm_Behaviour_Publishable`**

Adds a publication status on a `Nos\Orm\Model`. 2 modes are available:

- Yes / No state ;
- Publication depending on start / end dates (yes / no choice remains).

Configuration

property `Nos\Orm_Behaviour_Publishable::$`**`publication_state_property`**

Always required, both for the **yes/no** and the **date range** modes.

Column used to store the publication state. Its data type must be `int`:

- **0** means not published;
- **1** means always published;
- **2** means publication depends on a date range (when combining the 2 modes).

property `Nos\Orm_Behaviour_Publishable::$`**`publication_start_property`**

Required for **date range** mode. Column used to store the publication start date. Its data type must be `datetime`.

property `Nos\Orm_Behaviour_Publishable::$`**`publication_end_property`**

Required for **date range** mode. Column used to store the publication end date. Its data type must be `datetime`.

Methods

`Nos\Orm_Behaviour_Publishable::`**`published`**`()`
: **Returns** `true` or `false` depending on whether the item is currently published or not.

`Nos\Orm_Behaviour_Publishable::`**`planificationStatus`**`()`
: **Returns** `0` (not published), `1` (published) or `2` (scheduled).

`Nos\Orm_Behaviour_Publishable::`**`publicationStart`**`()`
: **Returns** the publication start date, MySQL format.

`Nos\Orm_Behaviour_Publishable::`**`publicationEnd`**`()`
: **Returns** the publication end date, MySQL format.

Other

This behaviour extends *Model->find()*.

You can use the `published` key in the `where` array. Appropriate conditions will be added, according to the configuration of the behaviour. Especially useful with the **date range** mode (and start / end dates).

Example

```php
<?php

// Yes/No state
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Publishable' => array(
                        'publication_state_property' => 'page_published',
                ),
        );
}

$page = Model_Page::find(1);

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

```php
<?php

// Date range mode (combined with Yes/No state)
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Publishable' => array(
                        'publication_state_property' => 'page_published',
                        'publication_start_property' => 'page_publication_start',
                        'publication_end_property'   => 'page_publication_end',
                ),
        );
}
```

Sharable

class `Nos\`**`Orm_Behaviour_Sharable`**

Adds the sharable behaviour on a `Nos\Orm\Model`.

Configuration

property `Nos\Orm_Behaviour_Sharable::$`**`data`**

Associative array of different types of data forming a `content nuggets`.

- Keys can be one of `DataCatcher` constants.
- Values are associative array.

  value A column name or a closure. See *Examples*.

  useTitle Help label indicate which property of item is use as default value.

  options Associative array of different options. Only for `DataCatcher::TYPE_URL`.

  possibles Associative array of possibles values. Only for `DataCatcher::TYPE_IMAGE`.

Methods

`Nos\Orm_Behaviour_Sharable::`**`get_default_nuggets`**`()`

Returns Array containing the default `content nuggets` of an item.

`Nos\Orm_Behaviour_Sharable::`**`get_catcher_nuggets`**`(`*$catcher = Model_Content_Nuggets::DEFAULT_CATCHER*`)`

Parameters

- **$catcher** (*string*) – `Data catchers` ID.

Returns The `Model_Content_Nuggets` of this item for the specified `data catcher`.

`Nos\Orm_Behaviour_Sharable::`**`get_sharable_property`**`(`*$property = null*, *$default = null*`)`

Parameters

- **$property** (*string*) – A property name.
- **$default** (*string*) – A default value if data property is null.

Returns `Orm_Behaviour_Sharable::$data` if `$property` is null. Otherwise, property value with default fallback.

`Nos\Orm_Behaviour_Sharable::`**`data_catchers`**`()`

Retrieves all the data catchers that can use the `content nugget` of this item (checks the `required_data` of a data catcher and).

returns All the valid `data catchers` for this item (keys are the `data catcher` names).

`Nos\Orm_Behaviour_Sharable::`**`possible_medias`**`()`

Retrieves all possible medias that can be associated with the item. Search for linked medias and images inserted in the WYSIWYGs.

returns Associative array of all `Model_Media`, `Model_Media` ID in keys, of item.

`Nos\Orm_Behaviour_Sharable::`**`get_nugget_content`**`(`*$catcher*`)`

Retrieves the personalised `content nugget` of a `data catcher`, merged with the default `content nugget` of the item.

param string $catcher `Data catchers` ID.

returns `Array` A `content nugget`.

Examples

A column for the default value

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

A closure which returns a default value

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

Real example From the `Monkey` example application. `config/model/monkey.config.php` (we're using a config file here, because we can't put functions in a class property):

```
<?php

return array(
    '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();
                                        },
                                ),
                        ),
                ),
        ),
);
```

Sortable

class `Nos\`**`Orm_Behaviour_Sortable`**

Makes a `Nos\Orm\Model` sortable.

Configuration

property `Nos\Orm_Behaviour_Sortable::$`**`sort_property`**

Required. Column name use for save sort index. Column must have type `double`.

property `Nos\Orm_Behaviour_Sortable::$`**`sort_order`**

`ASC` (default) ou `DESC`

Methods

`Nos\Orm_Behaviour_Sortable::`**`move_before`**`(`*$item*`)`

Parameters

- **$item** (*Model*) – Moves the current item before this one.

`Nos\Orm_Behaviour_Sortable::`**`move_after`**`(`*$item*`)`

Parameters

- **$item** (*Model*) – Moves the current item after this one.

`Nos\Orm_Behaviour_Sortable::`**`move_to_last_position`**`()`

Moves the current item to the last position (of its siblings).

Example

```php
<?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);
```

Tree

class `Nos\`**`Orm_Behaviour_Tree`**

Makes a `Nos\Orm\Model` behaves like a Tree.

An item can then have a parent and children (all of the same Model).

Configuration

property `Nos\Orm_Behaviour_Tree::$`**`parent_relation`**

Required. Name of the parent *relation*.

property `Nos\Orm_Behaviour_Tree::$`**`children_relation`**

Required. Name of the children *relation*.

property `Nos\Orm_Behaviour_Tree::$`**`level_property`**

Column used to store the item's depth inside the tree. Data type must be `int`.

Method

`Nos\Orm_Behaviour_Tree::`**`get_parent`**`()`

Returns Model parent item, if it exists, `null` otherwise.

`Nos\Orm_Behaviour_Tree::`**`set_parent`**`(`*$new_parent*`)`

Set a new parent for the item.

If the item is `twinnable` and if it exists in several contexts, all contexts will be moved synchronously.

Parameters

- **$new_parent** (*Model*) – New parent Model of the item (use `null` to remove the parent).

Throws `Exception` when:

- the item is moved in its own tree ;
- the item is `twinnable` and its parent does not exist in one of the contexts of the current item.

`Nos\Orm_Behaviour_Tree::`**`find_children`**(*$where = array(), $order_by = array(), $options = array()*)

Returns All direct children of item.

Children can be filter and / or sort by parameters.

This method use native method *Model->find()*.

`$options` parameter are passed to `->find()` like that:

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

`Nos\Orm_Behaviour_Tree::`**`find_children_recursive`**(*$include_self = true*)

Parameters

- **$include_self** (*boolean*) – If `true`, include current item in return.

Returns All children of item and their descendants.

`Nos\Orm_Behaviour_Tree::`**`find_root`**()

Returns First ascendant of item in tree or `null` if item has no parent.

Other

This behaviour extends *Model->find()*.

Add option to `where` array passed to method : you can use `parent` key as alias for search in `Orm_Behaviour_Tree::$parent_relation` relation.

Example

```php
<?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,
                ),
        );

}
```

Twinnable

class Nos\Orm_Behaviour_Twinnable

Extends `Nos\Orm_Behaviour_Contextable`.

It adds the ability to twin together different items with different contexts.

See Also:

Multi-Contexts.

See Also:

`Nos\Orm_Behaviour_Contextable` for configuration and methods.

Configuration

property `Nos\Orm_Behaviour_Twinnable::$`**`common_id_property`**

Required. Column used to store the common ID between twinned items. Data type must be `int`.

property `Nos\Orm_Behaviour_Twinnable::$`**`is_main_property`**

Required. Column used to store if the item is the main item among twin items. Data type must be `boolean`.

Methods

`Nos\Orm_Behaviour_Twinnable::`**`delete_all_context`**`()`

Removes all items twinned to the current item, including the current item itself.

`Nos\Orm_Behaviour_Twinnable::`**`is_main_context`**`()`

Returns `True` if item is the main among twin items.

`Nos\Orm_Behaviour_Twinnable::`**`find_context`**`(`*$context*`)`

Parameters

- **$context** (*mixed*) – Can be
  - Array of contexts ID.
  - `all`, to receive all contexts.
  - Context ID.
  - `main`, to receive main twin item.

Returns A twinned item, or an array of twinned items, `null` or `array()` if none.

`Nos\Orm_Behaviour_Twinnable::`**`find_main_context`**`()`

Returns The main item among the twins.

Alias for `->find_context('main')`.

`Nos\Orm_Behaviour_Twinnable::`**`find_other_context`**`(`*$filter = array()*`)`

Parameters

- **$filter** (*array*) – Array of contexts ID. If set, return only twin items which the context belongs to array `$filter`.

Returns Array of twin items, current item exclude.

`Nos\Orm_Behaviour_Twinnable::`**`get_all_context`**`()`

Returns Array of all twinned contexts, including the one of the current item.

`Nos\Orm_Behaviour_Twinnable::`**`get_other_context`**`(`*$filter = array()*`)`

Parameters

- **$filter** (*array*) – Array of contexts ID. If set, return only twinned contexts which belongs to array `$filter`.

Returns Array of all twinned contexts ID, excluding the one of the current item.

Example

```
<?php
class Model_Page extends \Nos\Orm\Model
{
        protected static $_behaviours = array(
                'Nos\Orm_Behaviour_Twinnable' => array(
                        'events' => array('before_insert', 'after_insert', 'before_save', 'after_dele
                        'context_property'      => 'page_context',
                        'common_id_property' => 'page_context_common_id',
                        'is_main_property' => 'page_context_is_main',
                        'invariant_fields'    => array(),
                ),
        );
}
```

Urlenhancer

class `Nos\`**`Orm_Behaviour_Urlenhancer`**

Used for `Nos\Orm\Model` displayed in the front-office by an *URL Enhancer*.

Configuration

property `Nos\Orm_Behaviour_Urlenhancer::$`**`enhancers`**

Required. Array of *enhancers* ID which can generate an URL for the item.

Listed *enhancers* must define a `get_url_model($item, $params)` method.

Methods

`Nos\Orm_Behaviour_Urlenhancer::`**`urls`**(*$params = array()*)

Parameters

- **$params** (*array*) –

  enhancer Specify *enhancer* ID. Restricts the search to the specified *enhancer*.

Returns

Associative array of all possibles URLs for this item

- key : `page_id::item_slug`. `item slug` is the URL part generate by *enhancer*.
- value : Absolute URL.

`Nos\Orm_Behaviour_Urlenhancer::`**`url`**(*$params = array()*)

Parameters

- **$params** (*array*) – See `Orm_Behaviour_Urlenhancer::urls`.

  canonical If `true`, return canonical URL of item.

  preview If `true`, return even unpublished URL of item.

Returns Absolute URL of item or `null` if item can't be displayed in front.

`Nos\Orm_Behaviour_Urlenhancer::`**`url_canonical`**(*$params = array()*)

Parameters

- **$params** (*array*) – See `Orm_Behaviour_Urlenhancer::url`.

Returns Absolute canonical URL of item or `null` if item can't be displayed in front.

Alias for `->url(array('canonical' => true))`.

If the item is `sharable`, returns the URL set in the `shared data (content nugget)`.

`Nos\Orm_Behaviour_Urlenhancer::`**`preview_url`**()

Returns Absolute canonical URL of item, even if it's not published, or `null` if item can't be displayed in the front-office.

Alias for `->url_canonical(array('preview' => true))`.

Example

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

Virtualname

class `Nos\`**`Orm_Behaviour_Virtualname`**

Adds a slug property to item.

The slug is automatically generated based on the `title_property` of the model if it is not specified.

> If `Orm_Behaviour_Virtualname::$unique` is set, `->save()` method can throw an `Exception` if slug already in use.

Configuration

property `Nos\Orm_Behaviour_Virtualname::$`**`virtual_name_property`**
Required. Column used to store the slug.

property `Nos\Orm_Behaviour_Virtualname::$`**`unique`**
`true`, `false` or `'context'` if uniqueness must be checked by context.

Methods

`Nos\Orm_Behaviour_Virtualname::`**`virtual_name`**`()`
Returns Item slug.

`Nos\Orm_Behaviour_Virtualname::`**`friendly_slug`**`(`*$slug*`)`
Parameters

- **$slug** (*string*) – A slug to clean.

Returns A clean slug, lowercase, without forbidden characters.

Example

```php
<?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',
                ),
        );
}
```

Virtualpath

class `Nos\`**`Orm_Behaviour_Virtualpath`**

Extends `Nos\Orm_Behaviour_Virtualname`.

Adds a virtual path to an item.

See Also:

`Nos\Orm_Behaviour_Virtualname` for configuration and methods.

Configuration

property `Nos\Orm_Behaviour_Virtualpath::$`**`virtual_path_property`**
Required. Column name use for save virtual path.

property `Nos\Orm_Behaviour_Virtualpath::$`**`extension_property`**
Required. If it's a `string`, it will be appended to the virtual path. If it's an `array`:

before String to add to extension start.

after String to add to extension end.

property Column name used to store the extension.

property `Nos\Orm_Behaviour_Virtualpath::$`**`extension_property`**
Required. Name of the parent *relation* use to generate the first part of the virtual path.

Methods

`Nos\Orm_Behaviour_Virtualpath::`**`virtual_path`**`(`*$dir = false*`)`

Parameters

- **$dir** (*boolean*) – If `true`, extension is replaced by a final `/`.

Returns Virtual path of the item.

`Nos\Orm_Behaviour_Virtualpath::`**`extension`**`()`

Returns Return extension part of the virtual path.

Example

```php
<?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',
                ),
        );
}
```

Printer friendly

Behaviour	Configuration	Methods
Publishable	• publication_bool_property	• ->published()
Sortable	• sort_property • sort_order	• ->move_before($item) • ->move_after($item) • ->move_to_last_position()
Contextable	• context_property • default_context	• ->get_context()
Twinnable	• common_id_property • is_main_property • context_property • default_context	• ->delete_all_context() • ->is_main_context() • ->find_context($context) • ->find_main_context() • ->find_other_context($filter) • ->get_all_context() • ->get_other_context($filter)
Tree	• level_property • parent_relation • children_relation	• ->get_parent() • ->set_parent($new_parent) • ->find_children($where = array(), $order_by = array(), $options = array()) • ->find_children_recursive($include_self) • ->find_root()
Urlenhancer	• enhancers	• ->urls($params = array()) • ->url($params = array()) • ->url_canonical($params = array()) • ->preview_url()
Virtualname	• virtual_name_property • unique	• ->virtual_name() • ::friendly_slug($slug)
Virtualpath	• virtual_name_property • virtual_path_property • unique • extension_property • parent_relation	• ->virtual_path($dir = false) • ->extension()
Sharable		

1.1.6 Classes

Nos

class `Nos\`**`Nos`**
Provides static methods useful in Novius OS.

Entry points

property `Nos\Nos::$`**`entry_point`**
The Novius OS entry point. Equal to one of the following constants.

constant `Nos\Nos::`**`ENTRY_POINT_ADMIN`**

constant `Nos\Nos::`**`ENTRY_POINT_FRONT`**

constant `Nos\Nos::`**`ENTRY_POINT_404`**

constant `Nos\Nos::`**`ENTRY_POINT_INSTALL`**

constant `Nos\Nos::`**`ENTRY_POINT_OIL`**

New in version 0.2.0.1.

::main_controller()

static `Nos\Nos::`**`main_controller`**
Returns Instance of the main *controller*.

::hmvc()

static `Nos\Nos::`**`hmvc`**(*$where*, *$args = null*)
Parameters

- **$where** (*mixed*) – Route for the request.
- **$args** (*array*) – The method parameters.

Executes an *HMVC* request.

::parse_wysiwyg()

static `Nos\Nos::`**`parse_wysiwyg`**(*$content*)
Parameters

- **$content** (*string*) – A `Nos\Model_Wysiwyg` content.

Returns Prepare wysiwyg content for display.

Replace enhancers by their content.

Replace `Model_Page` and `Model_Media` IDs by theirs URLs

Wysiwyg anchors are processed in case the href attribute begin with # : * If it begins with only one # then the href will be prefixed by the page url. * If if begins with two ## then it is transformed to only one #.

Front controller

class `Nos\`**`Controller_Front`**

Novius OS front-office controller

Use `Nos::main_controller` to retrieve its instance and call the methods.

Methods

`Nos\Controller_Front::`**`getContext`**`()`

Returns The current *context*.

`Nos\Controller_Front::`**`getContextUrl`**`()`

Returns Absolute URL of the current *context*.

`Nos\Controller_Front::`**`getPage`**`()`

Returns Current `Model_Page` displayed.

`Nos\Controller_Front::`**`getWysiwygName`**`()`

Returns Current `wysiwyg` ID processed.

`Nos\Controller_Front::`**`getUrl`**`()`

Returns Current absolute URL.

`Nos\Controller_Front::`**`getPageUrl`**`()`

Returns Relative (to `Controller_Front::getContextUrl`) URL of the current page.

`Nos\Controller_Front::`**`getEnhancedUrlPath`**`()`

Returns

Relative (to `Controller_Front::getContextUrl`) URL of the current *URL enhancer*.

`False` if no current *URL enhancer*.

Same that `Controller_Front::getPageUrl` ending with / instead of `.html`.

`Nos\Controller_Front::`**`getEnhancerUrl`**`()`

Returns

Part of the URL processed by the *URL enhancer*.

`False` if no current *URL enhancer*.

`Nos\Controller_Front::`**`setBaseHref`**`(`*$base_href*`)`

Parameters

- **$base_href** (*string*) – Sets a new `<base href="">` for the current HTML output.

`Nos\Controller_Front::`**`setTitle`**`(`*$title*, *$template = null*`)`

Parameters

- **$title** (*string*) – Set a new `title` for the current HTML.
- **$template** (*string*) – If set, use it to calculate the title. Placeholder `:title` will be replaced by `$title`.

`Nos\Controller_Front::`**`setMetaDescription`**`(`*$meta_description*`)`

Parameters

- **$meta_description** (*string*) – Set a meta description for the current HTML output.

`Nos\Controller_Front::`**`setMetaKeywords`**(*$meta_keywords*)

Parameters

- **$meta_keywords** (*string*) – Set a meta keywords for the current HTML output.

`Nos\Controller_Front::`**`setMetaRobots`**(*$meta_robots*)

Parameters

- **$meta_robots** (*string*) – Set a meta robots for the current HTML output.

`Nos\Controller_Front::`**`addMeta`**(*$meta*)

Parameters

- **$meta** (*string*) – A HTML meta tag to add in the current HTML output.

`Nos\Controller_Front::`**`addJavascript`**(*$url*, *$footer = true*)

Parameters

- **$url** (*string*) – URL of a JavaScript library to add in the current HTML output.
- **$footer** (*boolean*) – If `true`, add `script` at the end of HTML output. If `false`, add in the `<head>`.

`Nos\Controller_Front::`**`addJavascriptInline`**(*$js*, *$footer = true*)

Parameters

- **$js** (*string*) – Javascript code to add in the current HTML output.
- **$footer** (*boolean*) – If `true`, add at the end of HTML output. If `false`, add in the `<head>`.

`Nos\Controller_Front::`**`addCss`**(*$url*)

Parameters

- **$url** (*string*) – URL of a CSS file to add in the current HTML output.

`Nos\Controller_Front::`**`addCssInline`**(*$css*)

Parameters

- **$css** (*string*) – CSS code to add in the current HTML output.

`Nos\Controller_Front::`**`isPreview`**()

Returns Boolean, `true` if current page is requested in the preview mode.

`Nos\Controller_Front::`**`disableCaching`**()

Disable caching and cache retrieve of the current page.

`Nos\Controller_Front::`**`setCacheDuration`**(*$cache_duration*)

Parameters

- **$cache_duration** (*int*) – Set a new cache duration of the current cache saving.

`Nos\Controller_Front::`**`setStatus`**(*$status*)

Parameters

- **$cache_duration** (*int*) – Set a new response status of the current response. This status will be saved in cache.

`Nos\Controller_Front::`**`setHeader`**(*$name*, *$value*, *$replace = true*)

Add or replace a header to current response. Headers will be saved in cache.

Parameters

- **$name** (*string*) – The header name
- **$value** (*string*) – The header value
- **$replace** (*boolean*) – Whether to replace existing value for the header, will never overwrite/be overwritten when false

`Nos\Controller_Front::`**`getCustomData`**(*$item*, *$default = null*)
Returns a (dot notated) custom data of the current process.

Parameters

- **$item** (*string*) – Name of the custom data, can be dot notated.
- **$default** (*mixed*) – The return value if the custom data isn't found.

Returns The custom data or default if not found.

`Nos\Controller_Front::`**`setCustomData`**(*$item*, *$value*, *$cached = false*)
Sets a (dot notated) custom data to the current process.

Parameters

- **$item** (*string*) – A (dot notated) custom data key
- **$value** (*mixed*) – The custom data value
- **$cached** (*boolean*) – If custom data have to be cached

`Nos\Controller_Front::`**`sendContent`**(*$content*)
Replace the template by a specific content and stop treatments

Parameters

- **$content** (*mixed*) – The new content, can be a string or a View.

`Nos\Controller_Front::`**`addCacheSuffixHandler`**(*$handler*)
Add a cache suffix handler for the current page

Parameters

- **$handler** (*array*) – The cache suffix handler

Returns The cache instance if the cache path have changed, null otherwise.

FrontCache

class `Nos\`**`FrontCache`**
Manage front cache

Methods

static `Nos\FrontCache::`**`callHmvcUncached`**(*$uri*, *$args = array()*)
Write a HMVC call in cache for after cache execution.

Parameters

- **$uri** (*string*) – Route for the request.
- **$args** (*array*) – The method parameters.

See Also:

`Nos\Nos::hmvc`

static `Nos\FrontCache::`**`viewForgeUncached`**(*$file = null*, *$data = null*, *$auto_filter = null*)
Write a View forge in cache for after cache execution.

Parameters

- **$file** (*string*) – The view filename
- **$data** (*array*) – Array of values
- **$auto_filter** (*boolean*) – Set to true or false to set auto encoding

Attachment

class `Nos\`**`Attachment`**
Binds a file to an object.

Configuration

property `Nos\Attachment::$`**`attached`**
Required. The attached ID.

property `Nos\Attachment::$`**`dir`**
Required. Directory path, where the attachment is stored. Relative to `local/data/files`.

property `Nos\Attachment::$`**`alias`**
An URL alias to access the directory path.

property `Nos\Attachment::$`**`extensions`**
Array of valid files extensions.

property `Nos\Attachment::$`**`image`**
If `true`, accepts only files with an image extension. Same that `extensions` set to `array('jpg', 'gif', 'png', 'jpeg')`.

property `Nos\Attachment::$`**`check`**
Used it to make the attachment private. A callback function to check permissions against. It takes a single parameter: Attachement instance.

Methods

static `Nos\Attachment::`**`forge`**`(`*$attached*, *$config*`)`
Parameters

- **$attached** (*string*) – The attached ID.
- **$config** (*mixed*) – If is a `string`, use it to load configuration array. `Array` otherwise:

  dir `Attachment::$dir`

  alias `Attachment::$alias`

  extensions `Attachment::$extensions`

  image `Attachment::$image`

  check `Attachment::$check`

Returns A new `Nos\Attachment`.

`Nos\Attachment::`**`newFile`**`()`
Returns Get the new attachment file path if one, `false` if no file exists.

`Nos\Attachment::`**`path`**`()`
Returns Get the attachment file path or `false` if no file exists.

`Nos\Attachment::`**`filename`**`()`

Returns Get the attachment filename or `false` if no file exists.

`Nos\Attachment::`**`extension`**`()`

Returns Get the attachment extension or `false` if no file exists.

`Nos\Attachment::`**`isImage`**`()`

Returns `True` if the Attachment is an image, `false` otherwise.

`Nos\Attachment::`**`url`**`()`

Returns Get the attachment url or `false` if no file exists.

`Nos\Attachment::`**`urlResized`**`(`*$max_width = 0*, *$max_height = 0*`)`

Parameters

- **$max_width** (*array*) – Max width of the image.
- **$max_height** (*array*) – Max height of the image.

Returns Get the resized url for the Attachment or `false` if no file exists or it's not an image.

`Nos\Attachment::`**`set`**`(`*$file*, *$filename = null*`)`

Parameters

- **$file** (*array*) – File path
- **$filename** (*array*) – File name

Returns Set a new Attachment file.

Throws FuelCoreFileAccessException if new file have a not allowed extension.

`Nos\Attachment::`**`save`**`()`

Save a new Attachment file

`Nos\Attachment::`**`delete`**`()`

Delete the Attachment file

Example

```php
<?php

$attachment = \Nos\Attachment::forge('my_id', array(
        'dir' => 'apps'.DS.'myapps',
        'alias' => 'myapps-attachment',
        'extensions' => array('pdf'),
        'check' => 'check_attachment',
));

// It's for example
$_SESSION['user_connected'] = true;

function check_attachment($attachment) {
        return $GLOBALS['user_connected'];
}

try {
        $attachment->set('/path/a_doc.doc');
} catch (\Fuel\Core\FileAccessException $e) {
        // Exception will be throw, extension is doc, not a pdf.
}
```

```
$attachment->set('/path/a_pdf.pdf');
$attachment->save();

// Now file saved in local/data/files/apps/myapps/my_id/a_pdf.pdf

echo $attachment->url();
// Echo data/files/myapps-attachment/my_id/a_pdf.pdf

$_SESSION['user_connected'] = false;
// Now URL data/files/myapps-attachment/my_id/a_pdf.pdf return 404

$attachment->delete();
```

Tools_Context

class `Nos\`**`Tools_Context`**

Provides static methods to work with yours contexts, sites and languages.

See Also:

Multi-Contexts

::contexts()

static `Nos\Tools_Context::`**`contexts`**

Returns An array of all your valid contexts.

```
<?php
$contexts = \Nos\Tools_Context::contexts();
foreach ($contexts as $context_code => $context_urls) {
        // ....
}
```

::sites()

static `Nos\Tools_Context::`**`sites`**

Returns An array of all your valid sites. Each site has a `title` and an `alias`.

```
<?php
$sites = \Nos\Tools_Context::sites();
foreach ($sites as $site_key => $site_params) {
        $title = $site_params['title'];
        $alias = $site_params['alias'];
}
```

::locales()

static `Nos\Tools_Context::`**`locales`**

Returns An array of all your valid locales. Each locale has a `title` and a flag's code `flag`.

```
<?php
$locales = \Nos\Tools_Context::locales();
foreach ($locales as $locale_key => $locale_params) {
        $title = $locale_params['title'];
        $flag = $locale_params['flag'];
}
```

::defaultContext()

static `Nos\Tools_Context::`**`defaultContext`**

Returns The code of default context of your Novius OS instance.

```
<?php
$default_context_code = \Nos\Tools_Context::defaultContext();
```

::locale($container)

static `Nos\Tools_Context::`**`locale`**`(`*$container*`)`

Parameters

- **$container** (*string*) – A context code.

Returns Array of context's locale.

```
<?php
$locale = \Nos\Tools_Context::locale('main::en_GB');
$title = $locale['title'];
$code_flag = $locale['flag'];
```

::site($container)

static `Nos\Tools_Context::`**`site`**`(`*$container*`)`

Parameters

- **$container** (*string*) – A context code.

Returns Array of context's site.

```
<?php
$site = \Nos\Tools_Context::site('main::en_GB');
$title = $site['title'];
$alias = $site['alias'];
```

Tools_RSS

class `Nos\`**`Tools_RSS`**

Used to build a RSS feed.

Methods

::forge()

static `Nos\Tools_RSS::`**`forge`**(*$channel = array()*, *array $items = array()*)

Parameters

- **$channel** (*mixed*) – If it is a `string`, used as channel's `title`. Associative array otherwise:
- **$items** (*array*) – Associative array. Each key will be transformed into XML tag in a `<item />`.

Encoding Default `UTF-8`. Used for the XML encoding attribute.

Version Default `2.0`. Used for XML version attribute (`<rss>` tag).

You can define any other key, which will be transformed into XML tag in the `<channel />`

Returns A instance of `Tools_RSS`.

::set()

`Nos\Tools_RSS::`**`set`**(*$property*, *$value = null*)

Parameters

- **$property** (*mixed*) – A single `string` to set a channel property, or an associative array for multiple settings.
- **$value** (*mixed*) – If `$property` is a `string`, the value of the property.

Set one or multiple channel properties.

::set_items()

`Nos\Tools_RSS::`**`set_items`**(*array $items*)

Parameters

- **$items** (*array*) – Array of items.

Set a new array of items.

::add_item()

`Nos\Tools_RSS::`**`add_item`**(*array $item*)

Parameters

- **$item** (*array*) – An item.

Add a new item to the `$items` array.

::build()

`Nos\Tools_RSS::`**`build`**(*array $channel = array()*, *array $items = array()*)

Parameters

- **$channel** (*mixed*) –
- **$items** (*array*) –

Returns The XML description of the RSS

See `Tools_RSS::forge` for parameters.

The `pubDate` key can be a `Fuel\\Core\\Date` instance, or a string (date representation) or a timestamp.

Examples

```php
<?php
$rss = \Nos\Tools_RSS::forge('RSS title');
$rss->set_items(array(
        'title' => 'Item title',
        'link' => 'http://www.mydomain.com/item_url.html',
        'description' => '<p>A description of item </p>',
        'pubDate' => '2012-08-16',
        'author' => 'Me',
));
$xml = $rss->build();

$rss->set('subtitle', 'A subtitle for ma RSS');
echo $rss; // Call $rss->build() with magic method __ toString()
```

Tools_Url

class Nos\Tools_Url
Provides static methods for URL.

::page()

static Nos\Tools_Url::**page** (*$page_id*)
Parameters
- **$page_id** (*int*) – A `Model_Page` ID.

Returns URL of the specified page.

::context()

static Nos\Tools_Url::**context** (*$context*)
Parameters
- **$context** (*string*) – A context ID. See *Multi-Contexts*.

Returns Home URL of the specified context.

DataCatcher

class Nos\DataCatcher

constant Nos\DataCatcher::**TYPE_TITLE**

constant Nos\DataCatcher::**TYPE_URL**

constant Nos\DataCatcher::**TYPE_TEXT**

constant Nos\DataCatcher::**TYPE_IMAGE**

I18n

class `Nos\`**`I18n`**

Provides the translation related functions.

::setLocale()

static `Nos\I18n::`**`setLocale`**(*$locale*)

Configure the locale to use for translations.

Parameters

- **$locale** (*string*) – A valid locale (`en`, `en_GB`, `en_GB.utf-8` and `en_GB.utf-8@variant` are all valid).

::restoreLocale()

static `Nos\I18n::`**`restoreLocale`**

Restores the previous locale.

::load()

static `Nos\I18n::`**`load`**(*$file*)

Loads a dictionary for the current locale.

Parameters

- **$file** (*string*) – dictionary path.

::get()

static `Nos\I18n::`**`get`**(*$message*, *$default = null*)

Retrieves a translation from the last loaded dictionary.

Parameters

- **$message** (*string*) – The message to translate.
- **$default** (*string*) – The default text to return when the message is not found. Default value is the message itself.

::gget()

static `Nos\I18n::`**`gget`**(*$file*, *$message*, *$default = null*)

Retrieves a translation from a specific dictionary.

Parameters

- **$file** (*string*) – Which dictionary to look into.
- **$message** (*string*) – The message to translate.
- **$default** (*string*) – The default text to return when the message is not found. Default value is the message itself.

Warning: The dictionary must have been loaded manually before.

::dictionary()

static `Nos\I18n::`**`dictionary`**(*$files*)

Returns a closure that translate messages from a specific dictionary.

param string|array $files A dictionary file or list of dictionaries.

```php
<?php

// Retrieves an anonymous function
$dictionary = Nos\I18n::dictionary('mon_appli::common');

echo $dictionary('Translate this');
```

Migration

class `Nos\`**`Migration`**

Provides migration automation and methods useful for migrations.

Default usage

All you need is to declare the class in the right namespace.

```php
<?php

namespace Nos\Monkey\Migrations;

class Install extends \Nos\Migration
{
}
```

If the migration name is *001_install.php* it will try to execute *001_install.sql*.

If you want a more complex migration (e.g. update files), you can overload up and down functions.

->up()

`Nos\Migration::`**`up`**()

Tries to execute a sql file with same path (if migration filename is 001_install.php, it will try to execute 001_install.sql).

->down()

`Nos\Migration::`**`down`**()

Does nothing. Need to be overloaded if you want to support this operation.

::executeSqlFile($sql_file)

static `Nos\Migration::`**`executeSqlFile`**(*$sql_file*)

Parameters

- **$sql_file** (*string*) – Sql file location to be executed.

View

class View

Extends FuelPHP view class and provides new functionalities.

__construct($file, $data, $filter)

`View::__construct` (*$file = null*, *$data = null*, *$filter = null*)

Same specifications as in FuelPHP view class, except when you prefix the view name by *!*, view redirections are ignored.

::redirect($from, $to, $callback)

static `View::redirect` (*$from*, *$to*, *$callback = true*)

Parameters

- **$from** (*string*) – The view to redirect from.
- **$to** (*mixed*) – The view to redirect to or callback.
- **$callback** (*mixed*) – Callback

The function can redirect views: you can, for instance, make *View::forge('a')* return the content of view *b*.

Usages examples:

```php
<?php

View::redirect('a', 'b'); // redirects view 'a' to 'b'


View::redirect('a', 'b', function($data, $filter) { return false; }); // won't have any effect


View::redirect('a', 'b', function($data, $filter) { return false; });
View::redirect('a', 'c', function($data, $filter) { return true; });
// will redirect 'a' to 'c'


View::redirect('a', false, function($data, $filter) { return 'd'; })
// or
View::redirect('a', function($data, $filter) { return 'd'; })
// will redirect 'a' to 'd'
```

Permission

class Permission

static `Permission::check` (*$permission_name*, *$category_key = null*)

This is a helper to call `Nos\User\Model_User::check_permission` for the logged `Nos\User\Model_User`.

Parameters

- **$permission_name** (*string*) – Name of the permission to check against
- **$category_key** (*string*) – (optional) If the permission has categories, the category key to check against

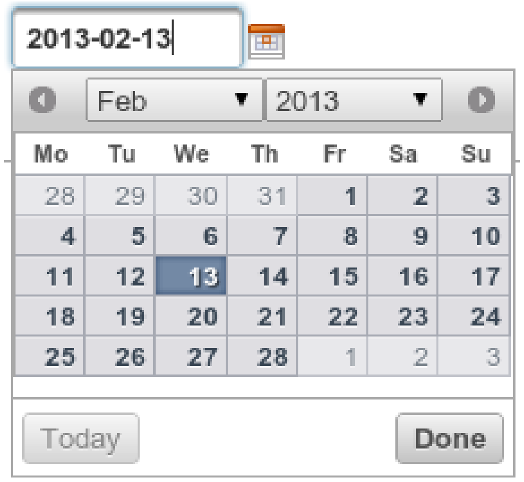
Returns `True` if the user has the required authorisation, `false` otherwise.

1.1.7 Renderers

Date Picker

class `Nos\`**`Renderer_Date_Picker`**

This renderer is used to pick a date.
It's based on jQuery UI Date Picker.



Configuration

property `Nos\Renderer_Date_Picker::$`**`wrapper`**
HTML string to wrap the `<input>` + the generated image to open the datepicker

property `Nos\Renderer_Date_Picker::$`**`datepicker`**
Options for the datepicker widget used to render the UI. See the jQuery UI documentation for all available options.

Default values below:

showOn both

buttonImage static/novius-os/admin/novius-os/img/icons/date-picker.png

buttonImageOnly true

autoSize true

dateFormat yy-mm-dd

showButtonPanel true

changeMonth true

changeYear true

showOtherMonths true

selectOtherMonths true

gotoCurrent true

firstDay 1

showAnim slideDown

Methods

`Nos\Renderer_Date_Picker::`**`renderer`**`(`*$renderer*`)`

Parameters

- **$renderer** (*Model*) – HTML attributes (`name`, `class`, `id`, `value`, etc.), with a special key `renderer_options`

Returns The <input> tag with JavaScript to initialise it

Displays a date picker in a standalone manner.

Example

Adding a date picker in a CRUD form configuration:

```php
<?php

return array(
    'label' => '',
    'renderer' => 'Nos\Renderer_Date_Picker',
    'renderer_options' => array(
        'datepicker' => array(),
        'wrapper' => '<div class="datepicker-wrapper"></div>',
    ),
);
```

Displaying a date picker:

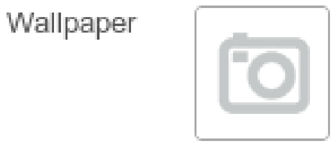
```php
<?php

echo Nos\Renderer_Date_Picker::renderer(array(
    'name' => 'my_date',
    'class' => 'some_class',
    'value' => '2013-02-13',
    'renderer_options' => array(
        'datepicker' => array(),
        'wrapper' => '<div class="datepicker-wrapper"></div>',
    ),
));
```

Media Selector

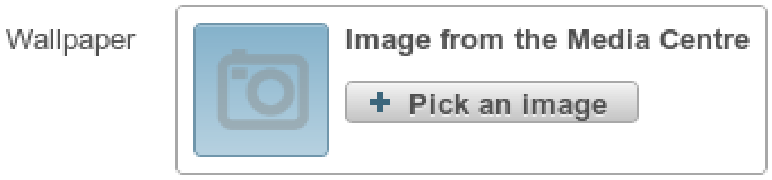
class `Nos\Media\`**`Renderer_Media`**

This renderer is used to pick a file from the media centre.

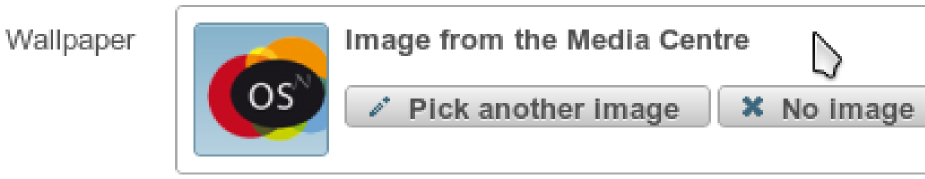
It's based on the jQuery UI input-file-thumb widget.



Default UI



Hover state UI



Hover state with a selected media UI

Configuration

property `Nos\Media\Renderer_Media::$`**`mode`**

Possible values are `image` (default) or `all`.

property `Nos\Media\Renderer_Media::$`**`inputFileThumb`**

Options for the inputFileThumb widget used to render the UI. See the inputFileThumb documentation for all available options.

Note: The `inputFileThumb.file` key will automatically be populated with the URL of the media if a `value` is provided in the renderer.

Methods

`Nos\Media\Renderer_Media::`**`renderer`**`(`*$renderer*`)`

Parameters

- **$renderer** (*Model*) – HTML attributes (`name`, `class`, `id`, `value`, etc.), with a special key `renderer_options`

Returns The <input> tag with JavaScript to initialise it

Displays a media selector in a standalone manner.

Example

Adding a media in a CRUD form configuration:

```php
<?php

return array(
    'label' => '',
    'renderer' => 'Nos\Media\Renderer_Media',
    'renderer_options' => array(
        'mode' => 'image',
            'inputFileThumb' => array(
            'title' => 'Title of the image',
        ),
    ),
);
```

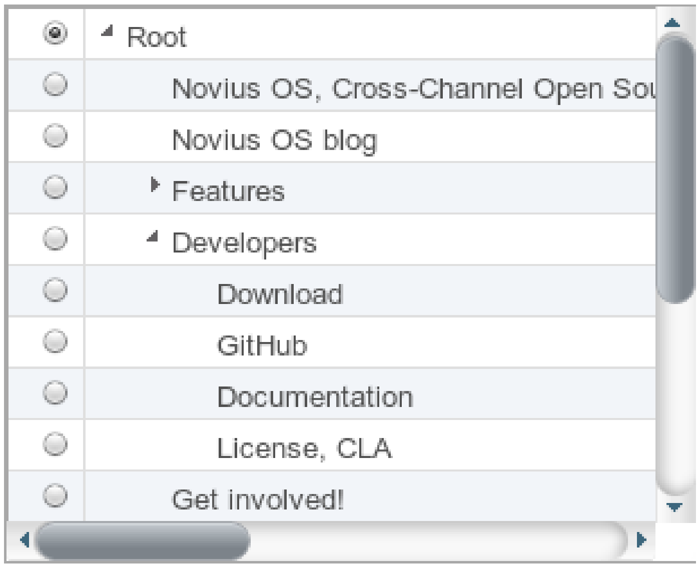
Displaying a media selector:

```
<?php

echo Nos\Media\Renderer_Media::renderer(array(
    'name' => 'my_image',
    'class' => 'some_class',
    'value' => 2, // ID of the previously selected media
    'renderer_options' => array(
        'mode' => 'image',
            'inputFileThumb' => array(
            'title' => 'Title of the image',
        ),
    ),
));
```

Page Selector

class `Nos\Page\`**`Renderer_Selector`**

This renderer is used to pick a page.
It displays a tree of the pages with radio buttons.



Configuration

property `Nos\Page\Renderer_Selector::$`**`input_name`**
Name for the radio input.

property `Nos\Page\Renderer_Selector::$`**`selected`**
`Array`

id Pre-selected page id (value).

property `Nos\Page\Renderer_Selector::$`**`treeOptions`**
`Array`

context Context of the pages displayed in the page selector.

property `Nos\Page\Renderer_Selector::$`**`height`**
Height of the renderer. Default is 150px.

property `Nos\Page\Renderer_Selector::$`**`width`**
Width of the renderer. Default is none.

Methods

`Nos\Page\Renderer_Selector::`**`renderer`**`(`*$renderer*`)`

Parameters

- **$renderer** (*Model*) – Array the attributes.

Returns The <input> tag with JavaScript to initialise it

Displays a page selector renderer in a standalone manner.

Example

Adding a page in a CRUD form configuration:

```php
<?php

return array(
    'label' => __('Location:'),
    'renderer' => 'Nos\Page\Renderer_Selector',
    'renderer_options' => array(
        'height' => '250px',
    ),
);
```

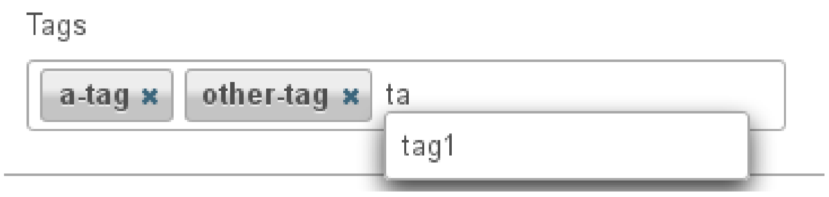
Displaying a media selector:

```php
<?php

echo Nos\Page\Renderer_Selector::renderer(array(
    'input_name' => 'my_page',
    'selected' => array(
        'id' => 2, // ID of the previously selected page
    ),
    'treeOptions' => array(
        'context' => 'main::en_GB',
    ),
    'height' => '250px',
));
```

Tag

class Nos**Renderer_Tag**

This renderer is used to add tags on an item.
It's based on jQuery UI Tag-it.



Configuration

Any options for the jQuery UI Tag-it widget. See the documentation for all available options.

Example

Adding a tags editor in a CRUD form configuration:

```
<?php

return array(
    'label' => '',
            'renderer' => 'Nos\Renderer_Tag',
            'renderer_options' => array(
                    'model'          => 'Model_Tag',
                    'label_column'   => 'tag_label',
                    'relation_name'  => 'tags'
            ),
);
```

Text

class Nos**Renderer_Text**
This renderer will show the value of the field as plain-text.

Example

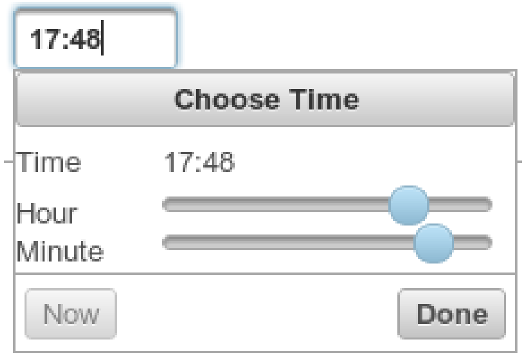
Adding a text in a CRUD form configuration:

```
<?php

return array(
    'label' => '',
    'renderer' => 'Nos\Renderer_Text',
);
```

Time Picker

class Nos\Renderer_Time_Picker

This renderer is used to pick a time.
It's based on jQuery UI Time Picker.



Configuration

All the jQuery UI Time Picker options can be used.

Methods

Nos\Renderer_Time_Picker::**renderer**(*$renderer*)

Parameters

- **$renderer** (*Model*) – HTML attributes (`name`, `class`, `id`, `value`, etc.), with a special key `renderer_options`

Returns The <input> tag with JavaScript to initialise it

Displays a time picker in a standalone manner.

Example

Adding a time picker in a CRUD form configuration:

```
<?php

return array(
    'label' => '',
    'renderer' => 'Nos\Renderer_Time_Picker',
    'renderer_options' => array(
        // jQuery UI Time Picker options
    ),
);
```

Displaying a time picker:

```
<?php

echo Nos\Renderer_Time_Picker::renderer(array(
    'name' => 'my_time',
    'class' => 'some_class',
    'value' => '17:48',
    'renderer_options' => array(
        // jQuery UI Time Picker options
    ),
));
```

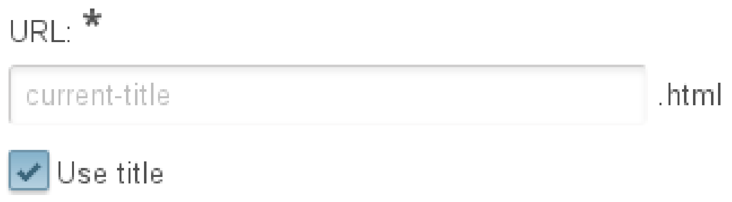
Virtual name

class `Nos\Renderer_Virtualname`

This renderer is used for modify an item slug.

See Also:

`NosOrm_Behaviour_Virtualname`



Example

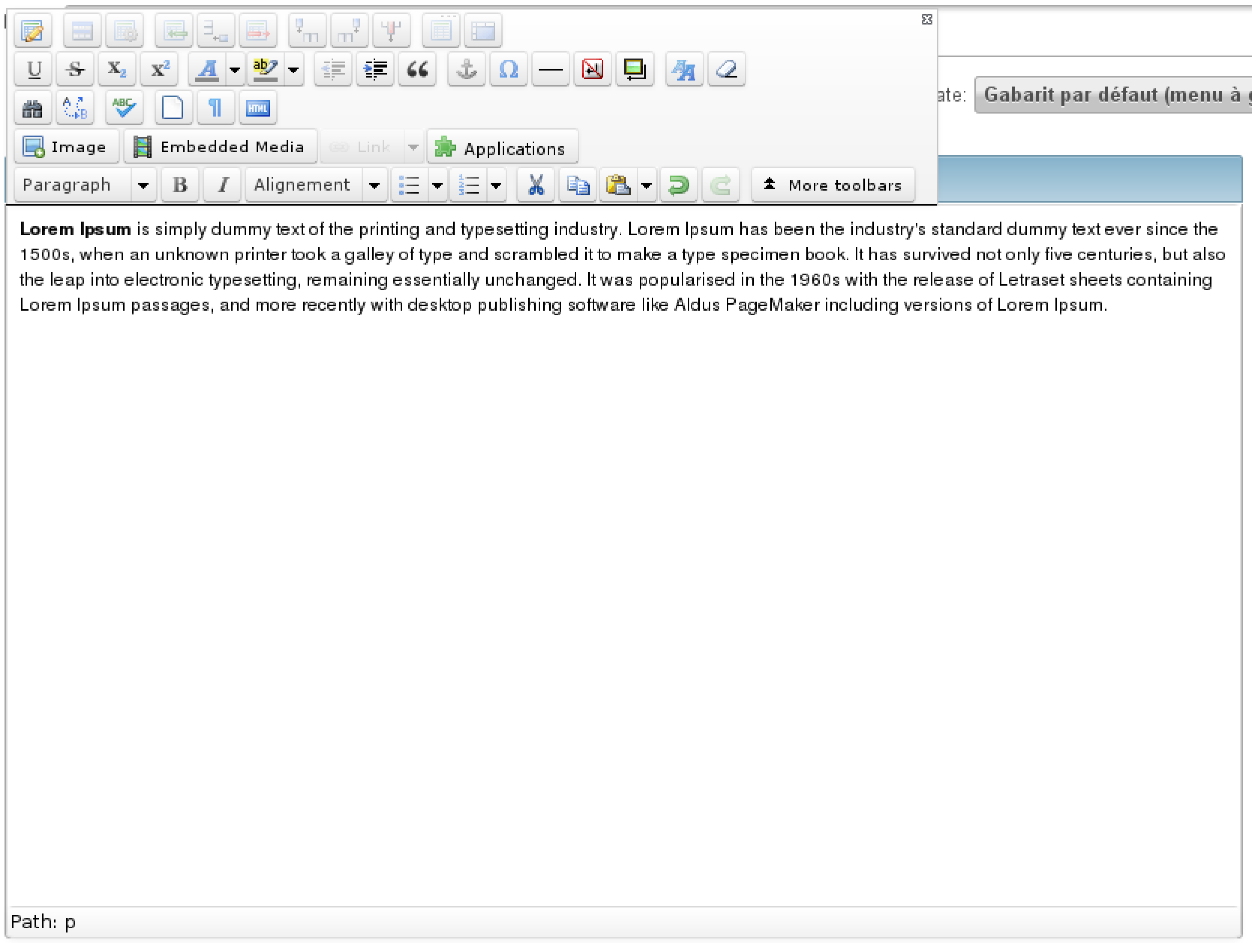
Adding a Virtual name in a CRUD form configuration:

```
<?php

return array(
    'label' => '',
    'renderer' => 'Nos\Renderer_Virtualname',
);
```

WYSIWYG

class `Nos\Renderer_Wysiwyg`

This renderer is used to display a WYSIWYG editor.
It's based on TinyMCE.



Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Path: p

Configuration

Any options for the TinyMCE WYSIWYG editor. See the TinyMCE documentation for all available options.

Methods

`Nos\Renderer_Wysiwyg::`**`renderer`**(*$renderer*)

Parameters

- **$renderer** (*Model*) – HTML attributes (`name`, `class`, `id`, `value`, etc.), with a special key `renderer_options`

Returns The <textarea> tag with JavaScript to initialise it

Displays a TinyMCE WYSIWYG editor in a standalone manner.

Example

Adding a WYSIWYG in a CRUD form configuration:

```
<?php
```

```
return array(
    'label' => '',
    'renderer' => 'Nos\Renderer_Wysiwyg',
    'renderer_options' => array(
            // TinyMCE options
    ),
);
```

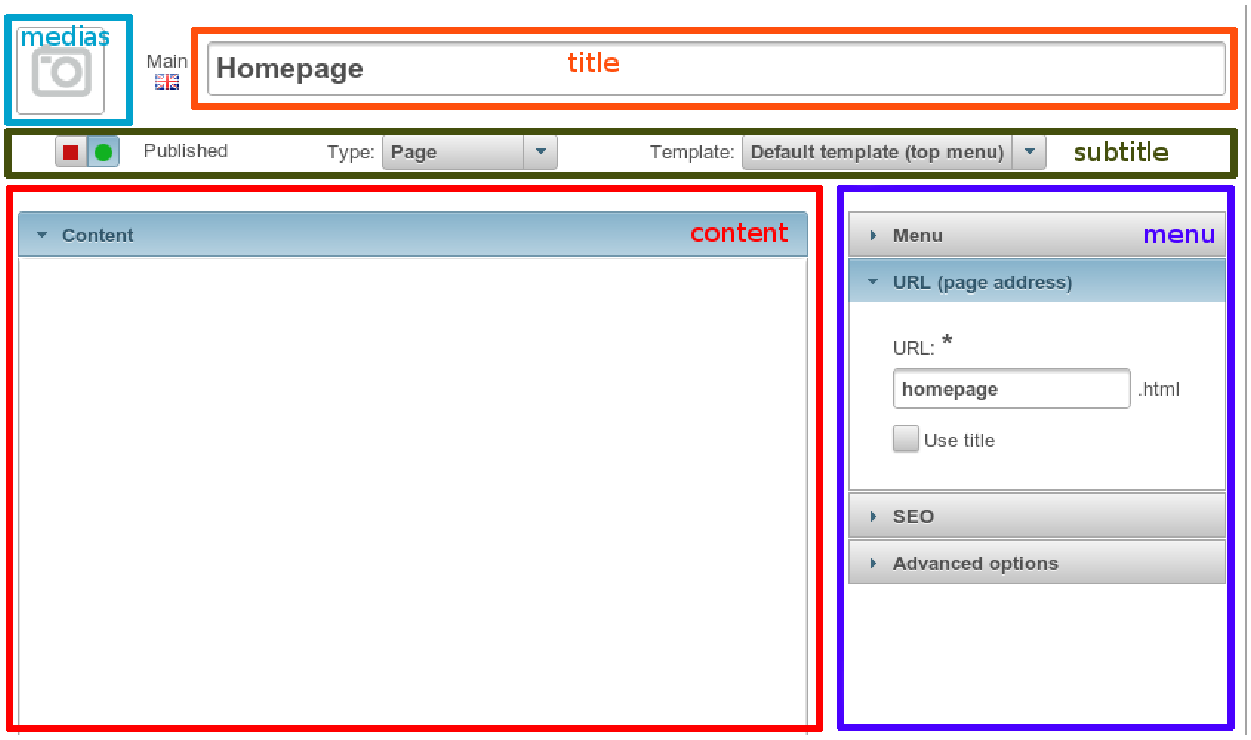
Displaying a WYSIWYG editor:

```
<?php

echo Nos\Renderer_Wysiwyg::renderer(array(
    'name' => 'my_wysiwyg',
    'value' => '<p>My contents</p>',
    'renderer_options' => array(
            // TinyMCE options
    ),
));
```

1.1.8 Views

Standard layout

View path: `nos::form/layout_standard`.



Params:

title Main fields at the top of the form.

medias Medias are shown at the left of the title.

large Default layout has spaces on the sides. If true, the form will use 100% of the width.

save Which field is the save button.

subtitle Fields under title.

content It has the same syntax as the **full** version of the *crud layout*.

menu Shown on the right. Data for the **accordion** view (see below). Optionally comes with an **simplified syntax**.

Standard syntax for the accordion:

```
array(
    'first_accordion' => array(
        'title' => __('My first accordion'),
        'fields' => array('field_1', 'field_2'),
    ),
    'second_accordion' => array(
        'title' => __('My second accordion'),
        'fields' => array('field_3', 'field_4'),
    ),
),
```

- if `title` is omitted, it will use the key as default value ;
- if `fields` is omitted, it will use the whole array as the list of fields.

(so you can't omit the `fields` if you set a `title`).

Simplified syntax for the accordion:

```
array(
    __('My first accordion') => array(
        'field_1',
        'field_2',
    ),
    __('My second accordion') => array(
        'field_3',
        'field_4',
    ),
),

// OR


array(
    __('My first accordion') => array(
        'fields' => array('field_1', 'field_2'),
    ),
    __('My second accordion') => array(
        'fields' => array('field_4', 'field_4'),
    ),
),
```

Expander

View path: `nos::form/expander`

See Also:

Wijmo Expander

Params:

title Expander's title.

options Options for the wijexpander.

content Either a plain *string*, a *callable*, or a *view* + *params* pair (array).

show_when_empty Optional. Default false. Should the expander be displayed if there is no content inside of it?

Accordion

View path: `nos::form/accordion`

See Also:

Wijmo Accordion

For each accordion, you can either set:

- a **view** + **params** pair ;
- or a **fields** list.

Params:

accordions A list of accordions

title Optional. Default value is *empty string*.

view Optional. Path to a view.

params Optional. Data for the `view`.

fields Optional. A list of field names.

field_template Optional. Template to wrap each field. Default value is `<p>{field}</p>`

header_class Optional. CSS classes for the `<h3>` tag.

content_class Optional. CSS classes for the `<div>` panel.

show_when_empty Optional. Default false. Should the panel be displayed if there is no content inside of it?

Fields

View path: `nos::form/fields`

Params:

begin String to display at the beginning.

end String to display at the end.

fields A list of field names.

callback Optional. Callback function used to render the field. By default, a field will be rendered by calling it's `build()` method.

show_when_empty Optional. Default false. Should the `begin` and `end` variables be displayed if there is no content inside of it?

Permissions standalone

View path: `nos::admin/permissions/standalone`

Params:

list A list of permissions

label Label of the permission.

checked `boolean` Initial state when the permission is not set (default: `false`).

permission_name (optional) Name of the permission. Default is to re-use this array key (see example below).

category_key (optional) Category of the permission

1.1.9 Events

Configuration

config|<path>

`config|<path>($config)`
A configuration file is loaded.

Parameters

- **&$config** (*array*) – The loaded array from the file

```php
<?php

// Listening to the event
Event::register_function('config|nos::controller/admin/page/page', function(&$config)
{
    // ...
}
// Triggering the event
$config = Config::load('nos::controller/admin/page/page', true);
```

Also works with absolute paths :

```php
<?php

// Listening to the event
Event::register_function('config|/data/www/novius-os/local/config/test.php', function(&$config)
{
    // ...
}
// Triggering the event (file must exists)
$config = Config::load('/data/www/novius-os/local/config/test.php', true);
```

config|<group>

`config|<group>($config)`
A configuration array is loaded.

Parameters

- **&$config** (*array*) – The loaded array

```
<?php

// Listening to the event
Event::register_function('config|group', function(&$config)
{
    // ...
}
// Triggering the event
Config::load(array(), 'group');
```

Front-office (website)

front.start

front.start($params)
An .html page is requested.
Parameters

- **$params** (*array*) –

  &$url `string` Current URL (without leading domain, with trailing .html)

  &$cache_path `string` Which entry should be checked / written in the cache

```
<?php

Event::register_function('front.start', function($params)
{
    $url =& $params['url'];
    $cache_path =& $params['cache_path'];
    // ...
});
```

front.parse_wysiwyg

front.parse_wysiwyg($html)
Additional processing on a WYSIWYG (HTML content).
Parameters

- **$html** (*string*) – HTML content, already pre-processed by the core

```
<?php

Event::register_function('front.parse_wysiwyg', function(&$content)
{
    // ...
});
```

front.display

front.display($html)

Additional processing on the page (HTML content).

Parameters

- **$html** (*string*) – HTML content of the page (will be written in the cache)

```php
<?php

Event::register_function('front.display', function(&$html)
{
    // ...
});
```

front.pageFound

front.pageFound($params)

Page to display have been found.

Parameters

- **$params** (*array*) –

  $page :php:class::*Nos\Page\Model_Page*

```php
<?php

Event::register('front.pageFound', function($params)
{
    $page = $params['page'];
    // ...
});
```

front.response

front.response($params)

Before that the response be sended.

Parameters

- **$params** (*array*) –

  &$content `string` The response body

  &$status `int` The HTTP response status for this response

  &$headers `array` HTTP headers for this response

```php
<?php

Event::register_function('front.response', function($params)
{
    $content =& $params['content'];
    $status =& $params['status'];
    $headers =& $params['headers'];
    // ...
});
```

front.404NotFound

front.404NotFound($params)

An .html page was requested, but not found.

Parameters

- **$params** (*array*) –

  $url `string` Current URL (without leading domain, with trailing .html)

```php
<?php

Event::register('front.404NotFound', function($params)
{
    $url = $params['url'];
    // ...
});
```

404 entry point

New in version 0.2.0.2.

404.start

404.start($params)

A inexistant file is requested. Can be media or attachment file.

Parameters

- **$params** (*array*) –

  &$url `string` URL requested (without leading domain)

```php
<?php

Event::register_function('404.start', function($params)
{
    $url =& $params['url'];
    // ...
});
```

404.mediaNotFound

404.mediaNotFound($url)

A inexistant media file is requested.

Parameters

- **$url** (*string*) – URL requested (without leading domain)

```php
<?php

Event::register('404.mediaNotFound', function($url)
{
    // ...
});
```

404.attachmentNotFound

404.attachmentNotFound($url)
A inexistant attachment file is requested.

Parameters

- **$url** (*string*) – URL requested (without leading domain)

```
<?php

Event::register('404.attachmentNotFound', function($url)
{
    // ...
});
```

404.end

404.end($url)
A inexistant file is requested. No attachment or media file matched the URL.

Parameters

- **$url** (*string*) – URL requested (without leading domain)

```
<?php

Event::register('404.end', function($url)
{
    // ...
});
```

Emails

email.before_send

email.before_send($email)
Before sending an email.

Parameters

- **$email** (*object*) – Email_Driver instance

```
<?php

Event::register('email.before_send', function($email)
{
    // ...
}
```

email.after_send

email.after_send($email)
After a mail was sent.

Parameters

- **$email** (*object*) – Email_Driver instance

```
<?php

Event::register('email.after_send', function($email)
{
    // ...
}
```

Forms application

noviusos_form::rendering

`noviusos_form::`**`rendering`**`(`*$params*`)`

Triggered by the enhancer before displaying the form to the user.

Allows to modify the `$fields` and the `$layout`. The `$item` (Model_Form instance) and `$enhancer_args` (label_position, etc.) variables are read-only.

Parameters

- **$params** (*array*) –

  &$fields Fields list

  - **label** Callback used to generate the label
    - **callback** `string` Callback function name
    - **args** `array` Callback function args
  - **field** Callback used to generate the field
    - **callback** `string` Callback function name
    - **args** `array` Callback function args
  - **instructions** Callback used to generate the instructions
    - **callback** `string` Callback function name
    - **args** `array` Callback function args
  - **new_row** `bool` Should the field be on a new row?
  - **new_page** `bool` Should the field be on a new page?
  - **width** `int` Column size (value between 1 and 4)
  - **item** `Model_Field` Field instance

  &$layout `string` Path to the view used to render the form

  $item `Model_Form` Form instance

```
<?php

Event::register_function('noviusos_form::rendering', function(array &$args)
{
    $fields =& $args['fields'];
    $layout =& $args['layout'];
    $form   =  $args['item']; // Instance of Nos\Form\Model_Form
```

```
        $enhancer_args = $args['enhancer_args'];

        // This is an exemple of what $layout contains
        $layout = 'noviusos_form::foundation';

        foreach ($fields as &$field) {

            // This is an example of what $field contains
            $field = array(
                'label' => array(
                    'callback' => array('Form', 'label'),
                    'args' => array('Label:', 'technical_id', array(
                        'for' => 'field_technical_id',
                    )),
                ),
                'field' => array(
                    'callback' => array('Form', 'input'),
                    'args' => array('field_name', 'field_value', array(
                        'id'          => 'field_technical_id',
                        'class'       => 'field_technical_css',
                        'title'       => 'Label:',
                        'placeholder' => 'Label:',
                        'required'    => 'required',
                    )),
                ),
                'instructions' => array(
                    'callback' => 'html_tag',
                    'args' => array('p', array('class' => 'instructions'), 'Instructions for the use
                ),
                'new_row' => true,
                'new_page' => true,
                'width' => 4,
                'item' => $item, // Instance of Nos\Form\Model_Field
            );
        }

        // ...
    }
```

noviusos_form::rendering.<virtual_name>

Same as `noviusos_form::rendering`, but only triggered for a form with the specified `<virtual_name>`.

noviusos_form::data_validation

warning ! This function must return an array containing the detected validation errors.

`noviusos_form::`**`data_validation`**(*&$data*, *$form*)

Additional data validation after submitting a form from the Form application.

Parameters

- **&$data** (*array*) – Received data (roughly $_POST)
- **$form** (*object*) – `Model_Form` Form instance

Return array List of validation errors

```php
<?php

Event::register_function('noviusos_form::data_validation', function(&$data, $form) {

    $errors = array();
    // This will mark all fields as error
    foreach ($data as $name => $value) {
        $errors[$name] = '{{label}}: `{{value}}' non valid.';
    }
    return $errors;
});
```

The messages can contain the `{{label}}` and `{{value}}` *placeholders* (they will be replaced appropriately).

noviusos_form::data_validation.<virtual_name>

Same as `noviusos_form::data_validation`, but only triggered for a form with the specified `<virtual_name>`.

noviusos_form::before_submission

`noviusos_form::`**`data_validation`**`(`*&$data*, *$form*`)`

Before saving the answer in the database

Parameters

- **&$data** (*array*) – Received data (to save in DB)
- **$form** (*object*) – `Model_Form` Form instance
- **$enhancer_args** (*array*) – Enhancer configuration

Return bool false to prevent saving the answer in the database

```php
<?php

Event::trigger_function('noviusos_form::before_submission', array(&$data, $form, $enhancer_args)

    // You can alter $data before saving them into the database

    // Or you can return 'false' if you don't want the answer to be saved in the database
    return false;
});
```

noviusos_form::before_submission.<virtual_name>

Same as `noviusos_form::before_submission`, but only triggered for a form with the specified `<virtual_name>`.

noviusos_form::after_submission

`noviusos_form::`**`after_submission`**`(`*&$answer*, *$enhancer_args*`)`

After the answer has been created (not saved in the DB yet)

Parameters

- **&$answer** (*object*) – `Model_Answer` Answer instance
- **$enhancer_args** (*array*) – Enhancer configuration

```
<?php

Event::trigger_function('noviusos_form::after_submission', array(&$answer, $enhancer_args) {

    // ...
});
```

noviusos_form::after_submission.<virtual_name>

Same as `noviusos_form::after_submission`, but only triggered for a form with the specified `<virtual_name>`.

1.1.10 FuelPHP

OrmModel

Native FuelPHP ORM Model. The M of MVC.
See http://fuelphp.com/docs/packages/orm/creating_models.html

Controller

The C of MVC.
See http://fuelphp.com/docs/general/controllers/base.html

Relations

FuelPHP ORM mechanism to link Model between them.
See http://fuelphp.com/docs/packages/orm/relations/intro.html

Observers

FuelPHP ORM mechanism to add event on Model.
See http://fuelphp.com/docs/packages/orm/observers/intro.html

Environments

FuelPHP manages environments natively.
See http://fuelphp.com/docs/general/environments.html

HMVC

A FuelPHP concept.
See http://fuelphp.com/docs/general/hmvc.html

has_many

A native *FuelPHP ORM relation*.
See http://fuelphp.com/docs/packages/orm/relations/has_many.html

belongs_to

A native *FuelPHP ORM relation*.
See http://fuelphp.com/docs/packages/orm/relations/belongs_to.html

many_many

A native *FuelPHP ORM relation*.
See http://fuelphp.com/docs/packages/orm/relations/many_many.html

Model->find()

Native method of FuelPHP ORM Model.
See http://fuelphp.com/docs/packages/orm/crud.html#read

1.2 Javascript API

1.2.1 $ functions

$

In this documentation, `$` means the jQuery object itself.

```
$.nosNotify('message');
```

Functions

nosDataReplace

`$.nosDataReplace` (*object*, *data*)

Replaces placeholder defined by the `{{placeholder}}` pattern in a string or JSON.

Arguments

- **object** (*mixed*) – A string or a JSON, where to search placeholder to replace.
- **data** (*JSON*) – A JSON, `placeholder => replacement value`.

Returns Initial object with placeholders replaced.

```
$.nosDataReplace(obj, data);

$.nosDataReplace('exemple {{foo}}', {foo : 'bar'});
// returns 'exemple bar'

$.nosDataReplace({
        astring: 'example {{foo}}',
        json: {
                string: 'sample {{bar}}',
        }
},
{
        foo : 'bar',
        bar : 'foo'
});
//returns {
//      astring: 'example bar',
//      json: {
//              string: 'sample foo',
//}
```

nosNotify

`$.nosNotify` (*message*[, *type*])

Displays notification(s) to the user. This is a wrapper of the jQuery plugin Pines Notify.

Arguments

- **message** (*mixed*) – `string` The message to display or a JSON for special configuration.
- **type** (*string*) – Type of the message. Can be `notice` (default), `info`, `success` or `error`.

```
$.nosNotify('message');

$.nosNotify('message', 'error');

$.nosNotify({
        title: 'It\'s working',
        type: 'success'
        sticker: false, // Don't provide a button for the user to manually stick the notice.
        hide: false, // Don't remove the notification after a delay.
});
```

nosUIElement

`$.nosUIElement`(*element*[, *data*])

Arguments

- **element** (*JSON*) – JSON definition of the element to create.

  type `string`. `button` (default) or `link`. See `$container.nosFormUI()` for buttons `data`, those of links are almost the same.

  label `string`. Element label.

  action `{}`. Action bound to the click event. See `$container.nosAction()`.

  bind `{}`. Event(s) bound to the element. See $().bind().

  disabled `boolean` or `string`. If `true` or `string`, the element is disabled. If it is a `string`, it will fill the element's title.

  menu `{}`. If set, binds a context menu to element.

    menus `[{}]`. Array containing each menu line.

      action `{}`. Action bound to the click event of the menu line. See `$container.nosAction()`.

      content `string`. HTML content of the menu line.

      label `string`. Label of the menu line.

      icon Icon name (See icons name of jQuery UI) without the `ui-icon-` prefix.

      iconClasses CSS classes of icon.

      iconUrl Icon URL.

    options `{}`. Settings for Wijmo widget wijmenu.

- **data** (*JSON*) – Data attached to element and passed to action. See `$container.nosAction()`.

Returns A DOM element detach from DOM.

```
$.nosUIElement({
        type: 'button',
        label: 'A button'
},
{
        id: 5
        foo: 'bar'
});
```

Note: `content` and `label` are exclusive. You don't need both. Same goes for the `icon` (either use a name, CSS classes or an URL).

1.2.2 $container functions

`$container`
In this documentation, `$container` means a jQuery collection of DOM elements.

```
$('#id').nosTabs('open', {
        url: 'url',
        iframe: false,
        label: 'Title',
        labelDisplay: true,
        iconUrl: 'icon.png',
        iconSize: 16
});
```

Function with sub-functions Many functions of the Novius OS Javascript API have sub-functions. The first parameter is the name of a sub-function. If this parameter is omitted, the default sub-function is called.

```
$('#id').nosTabs('open', {
        url: 'url',
        iframe: false,
        label: 'Title',
        labelDisplay: true,
        iconUrl: 'icon.png',
        iconSize: 16
});

// This call do the same thing that the previous
$('#id').nosTabs({
        url: 'url',
        iframe: false,
        label: 'Title',
        labelDisplay: true,
        iconUrl: 'icon.png',
        iconSize: 16
});
```

Functions

nosAction

`$container.`**`nosAction`**(*action*[, *data*])

Executes an action

Arguments

- **action** (*JSON*) – JSON defines the action to execute. JSON must have an `action` key, can be :

  nosTabs `nosAction.nosTabs`

  nosDialog `nosAction.nosDialog`

  confirmationDialog `nosAction.confirmationDialog`

  nosAjax `nosAction.nosAjax`

  window.open `nosAction.windowOpen`

  document.location `nosAction.documentLocation`

  nosMediaVisualise `nosAction.nosMediaVisualise`

  dialogPick `nosAction.dialogPick`

- **data** (*JSON*) – JSON contextual data. Used to replace placeholder in `action` by calling `$.nosDataReplace()`.

Actions list

- nosTabs
- nosDialog
- confirmationDialog
- nosAjax
- window.open
- document.location
- nosMediaVisualise
- dialogPick

nosTabs

`nosAction.`**`nosTabs`**

Action `nosTabs`

Method Any sub-function name of `$container.nosTabs()`.

Tab First parameter passed to sub-function of `$container.nosTabs()`.

Dialog Second parameter passed to sub-function of `$container.nosTabs()`.

```
$(domContext).nosAction({
        action: 'nosTabs',
        method: 'add',
        tab: {
                url: 'path/url',
                label: 'A title',
                iconUrl: 'url/of/icon.png'
        },
        dialog: {
                width: 800,
                height: 400
        }
});
```

nosDialog

`nosAction.`**`nosDialog`**

Action `nosDialog`

Dialog Second parameter passed to `nosDialog.open()`.

```
$(domContext).nosAction({
        action: 'nosDialog',
        dialog: {
                ajax: true,
                contentUrl: 'path/url/,
                title: 'A title',
                width: 500,
                height: 200
        }
});
```

confirmationDialog

`nosAction.`**`confirmationDialog`**

A special form of `nosAction.nosDialog` for confirmation.

Action `confirmationDialog`

Dialog Second parameter passed to `nosDialog.open()`.

```
$(domContext).nosAction({
        action: 'confirmationDialog',
        dialog: {
                contentUrl: 'path/url/,
                title: 'A title'
        }
});
```

nosAjax

`nosAction.`**`nosAjax`**

Action `nosAjax`

Params Settings of `$container.nosAjax()`.

```
$(domContext).nosAction({
        action: 'nosAjax',
        params: {
                url: 'path/url',
                method: 'POST',
                data: {
                        id: '{{_id}}'
                )
        }
}, {
        _id: 5
});
```

window.open

nosAction.**windowOpen**

Open a new browser window.

Action `window.open`

Url URL of the new window.

```
$(domContext).nosAction({
        action: 'window.open',
        url: 'path/url'
});
```

document.location

nosAction.**documentLocation**

Redirects the browser window to a new URL.

Action `document.location`

Url The new URL.

```
$(domContext).nosAction({
        action: 'document.location',
        url: 'path/url'
});
```

nosMediaVisualise

nosAction.**nosMediaVisualise**

This action has no parameters. It only depends on the data passed with action. See `$.nosMediaVisualise()`.

Action `nosMediaVisualise`

```
$(domContext).nosAction({
        action: 'nosMediaVisualise'
}, {
        path: 'url/of/media/',
        image: true
});
```

dialogPick

nosAction.**dialogPick**

Action `dialogPick`

Event Name of the event to trigger.

```
$(domContext).nosAction({
        action: 'dialogPick',
        'event' => 'event_name'
});
```

nosAjax

$container.**nosAjax**(*options*)

Performs an asynchronous HTTP (AJAX) request. This is a wrapper of jQuery.ajax().

Arguments

- **options** (*JSON*) – JSON that configure the AJAX request. Same that `jQuery.ajax()` with some differences:
  - Some defaults options.

- Callback functions `success` and `error` are monkey-patched to execute defaults operations.
  - Function `$container.nosAjaxSuccess()` is automatically called if the request succeeds and return type is JSON.
  - Function `$container.nosAjaxError()` is automatically call if the request fails.

```
$(domContext).nosAjax({
        dataType: 'json', // datatype is default 'json'.
        type: 'POST', // The request is made in POST by default.
        data: {}
});
```

nosAjaxSuccess

`$container.`**`nosAjaxSuccess`**(*options*)

Process JSON of a succeeded AJAX request.

Arguments

- **options** (*JSON*) –

  notify A `string` or `[string]`. Use for call `$.nosNotify()`.

  error A `string` or `[string]`. Use for call `$.nosNotify()` with `error` for notification type.

  action A `string` or `[string]`. Use for call `$container.nosAction()`.

  closeDialog `Boolean`. If `true`, call `nosDialog.close()`.

  closeTab `Boolean`. If `true`, call `nosTabs.close()`.

  replaceTab `{}`. Use to call `nosTabs.update()`.

  redirect `string`. Redirect browser window to this URL.

  dispatchEvent Use to call `$container.dispatchEvent()`.

  internal_server_error Display error and backtrace in the browser console.

nosAjaxError

`$container.`**`nosAjaxError`**(*jqXHR*, *textStatus*)

Process a failed AJAX request.

Display the reconnection popup if the error comes to an end of the authentication session.

Display a notification otherwise.

Arguments

- **jqXHR** (*jqXHR*) – An XMLHttpRequest object.
- **textStatus** (*string*) – Text status of the AJAX request.

nosDialog

`$container.nosDialog([method[, options[, ...]]])`

Manages back-office's popup. This is a *function with sub-functions*.

Arguments

- **method** (*mixed*) – The sub-function name, `open` (default) or `close`. If omitted this is the first parameter of the default sub-function `nosDialog.open()`.
- **options** (*mixed*) – Parameters of the sub-function.

nosDialog('open')

`nosDialog.open(dialog)`

Opens a popup. This is a wrapper of Wijmo widget Dialog.

A popup can be created in three ways:

- From an existing DOM element.
- From an URL loaded into an `iframe`.
- Form an URL loaded into a `<div>` from an AJAX request.

Catch events dispatched by `$.nosDispatchEvent()`.

Arguments

- **options** (*JSON*) – JSON that configures the popup. Same as `$.wijdialog()` with some differences:

  Some defaults options:

  width Container width minus 200 pixels.

  height Container height minus 100 pixels.

  modal True

  captionButtons Buttons `pin`, `refresh`, `toggle`, `minimize` and `maximize` are hidden.

  Additional options:

  destroyOnClose `boolean`. Destroys the popup when it's closed. Default `true`.

  ajax `boolean`. If `true`, `contentUrl` is loaded using AJAX rather than using an `iframe`. Default `true`.

  ajaxData `{}`. Data passed to the AJAX request (if `ajax` is `true`).

```
// Popup containing the HTML result of the AJAX request on contentUrl
$(domContext).nosDialog('open', {
        contentUrl: 'path/url',
        ajaxData: {
                foo: 'bar'
        },
        title: 'title of the popup',
        height: 400,
        width: 700
});

// Same as previous, without first parameter, open is the default sub-function
$(domContext).nosDialog({
        contentUrl: 'path/url',
        ajaxData: {
                foo: 'bar'
        },
        title: 'title of the popup',
        height: 400,
        width: 700
});

// Popup containing an iframe width contentUrl href
$(domContext).nosDialog({
        iframe: true
        contentUrl: 'path/url',
        title: 'title of the popup',
        height: 400,
        width: 700
});

// Popup containing <div> with the id 'id_de_div'
$('#id_de_div').nosDialog({
        title: 'title of the popup',
        height: 400,
        width: 700
});
```

nosDialog('close')

`nosDialog.`**`close`**`()`

Closes the current popup dialog (i.e. from the current js:data::*DOM container <$container>*).

```
$(domContext).nosDialog('close');
```

nosFormUI

`$container.`**`nosFormUI`**`()`

Will perform UI enhancements on DOM elements on all the children of js:data::*$container* using Wijmo and jQuery UI widgets.

Element with a `.notransform` CSS class will be left unchanged.

Input text Using the wijtextbox widget.
Select Using the wijdropdown widget.
Checkbox Using the wijcheckbox widget.
Radio Using the wijradio widget.
Expander

Elements with the `.expander` CSS class using wijexpander widget.

You can set options with the `data-wijexpander-options` HTML attribute (JSON for additional settings).

Accordion Elements with CSS class `.accordion` using wijaccordion widget.
Submit

Using the button widget. | You can set options with the `data-` HTML attributes (additional settings).

red Makes the button red.

icons `{}`. Define icons for the button.

icon Defines the left icon using a name (See icons names of jQuery UI).

iconClasses Defines the left icon CSS classes for left icon.

iconUrl Define URL of the left icon.

```
$(domContext).nosFormUI();
```

nosFormAjax

`$container.`**`nosFormAjax`**`()`

Will use the jquery-form plugin to submit any form inside js:data::*$container* using Ajax rather than the native browser action.

The default response data type is `json`, and the `success` and `error` callbacks functions will call `$container.nosAjaxSuccess()` and `$container.nosAjaxError()`.

```
$(domContext).nosFormAjax();
```

nosFormValidate

`$container.`**`nosFormValidate`**`()`

Will use the jquery-validation plugin to perform inline validation on any form inside js:data::*$container*.

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 form's standard layout.

```
$(domContext).nosFormValidate({});
```

nosMedia

`$container.`**`nosMedia`**`([`*options*`])`

Transforms an `<input type="hidden">` element into a media selector UI in Media Center using plugin inputFileThumb.

Arguments

- **options** (*JSON*) – Optional options

  mode `string`. Can be `image` (default) or `all`.

  inputFileThumb `{}`. options for `inputFileThumb` plugin.

```
$(:input).nosMedia();

$(:input).nosMedia({
        mode: 'image',
        inputFileThumb: {
                title: 'a title'
        }
});
```

See Also:

Media Selector.

nosMediaVisualise

`$.nosMediaVisualise(media)`

Displays a media, in a popup for images or in a new browser window for other documents (like PDF).

Arguments

- **media** (*JSON*) – Media parameters

  path `string`. Media URL.

  image `boolean`.

```
$.nosMediaVisualise({
        path: 'url/of/media/',
        image: true
});
```

nosOnShow

`$container.nosOnShow([method[, options[, ...]]])`

Special API which delays the rendering of UI elements when they become visible. A lot of UI elements don't initialise correctly when they are hidden (they can't calculate sizes properly).

This is a *function with sub-functions*.

Arguments

- **method** (*mixed*) – The sub-function name, `show` (default), `one` or `bind`. If omitted, this is the first parameter of the default sub-function `nosOnShow.show()`.
- **options** (*mixed*) – Parameters of the sub-function.

nosOnShow('show')

`nosOnShow.show()`

Triggers all functions bounded with `nosOnShow.bind()` for any children of `$container`.

Warning: You have to actually show the element before calling this function.

```
$(domContext).show().nosOnShow();

// Or
$(domContext).show().nosOnShow('show');
```

nosOnShow('one')

`nosOnShow.one(callback)`

Binds a callback function which will be called only the one time (at the first display).

Arguments

- **callback** (*function*) – A callback function.

```
$(element).nosOnShow('one', function() {
        $(this).widget();
});
```

nosOnShow('bind')

`nosOnShow.bind(callback)`

Binds a callback function which will be called each time that the element becomes visible.

Arguments

- **callback** (*function*) – A callback function.

```
$(element).nosOnShow('bind', function() {
        $(this).widgetRefresh(); // widgetRefresh don't exist, it's an example.
});
```

nosTabs

`$container.nosTabs([method[, options[, ...]]])`

Manages back-office's tabs. This is a *function with sub-functions*.

Arguments

- **method** (*mixed*) – The sub-function name, `open` (default), `add`, `close`, `update` or `current`. If omitted, this is the first parameter of the default sub-function `nosTabs.open()`.
- **options** (*mixed*) – Parameters of the sub-function.

nosTabs('open')

`nosTabs.open(tab[, dialogOptions])`

Open a new tab or re-open an existing tab if it has the same URL.

Arguments

- **tab** (*mixed*) – JSON definition of tab.

  url Required. Tab URL.

  iframe If true, open tab in an iframe. Default `false`.

  label Tab label.

  labelDisplay If false, don't display the label, only the icon. Default `true`.

  iconUrl Icon URL.

  iconSize Icon size in pixel (square icon). Default 16.
- **dialogOptions** (*JSON*) – Within a popup, a tab will rather open another new popup instead (by calling `$container.nosDialog()`). This parameter set options for `$container.nosDialog()`.

```
$(domContext).nosTabs('open', {
        url: 'path/url',
        iframe: false,
        label: 'title',
        labelDisplay: true,
        iconUrl: 'path/icon.png',
        iconSize: 16
});

// Call simplified
$(domContext).nosTabs({
        url: 'path/url',
        iframe: false,
        label: 'title',
        labelDisplay: true,
        iconUrl: 'path/icon.png',
        iconSize: 16
});
```

nosTabs('add')

`nosTabs.add(tab[, dialogOptions, position])`

Adds a new tab, even if an existing has the same URL.

Arguments

- **tab** (*mixed*) – JSON definition of tab.

  url Required. Tab URL.

  iframe If true, open tab in an iframe. Default `false`.

  label Tab label.

  labelDisplay If false, don't display the label, only the icon. Default `true`.

  iconUrl Icon URL.

  iconSize Icon size in pixel (square icon). Default 16.
- **position** (*string*) – Position of the new tab. Can be `end`, `before` or `after` the current tab of the js:data::*jQuery $container <$container>*.
- **dialogOptions** (*JSON*) – If DOM element in jQuery container is in popup, open a new popup by calling `$container.nosDialog()` instead of a tab. This parameter set options for `$container.nosDialog()`.

```
$(domContext).nosTabs(
        'add',
        {
                url: 'path/url',
                iframe: false,
                label: 'title',
                labelDisplay: true,
                iconUrl: 'path/icon.png',
                iconSize: 16
        },
        'end'
);
```

nosTabs('close')

`nosTabs.close()`

Close current tab (compared to the tab where is the DOM element in jQuery container).

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

nosTabs('update')

`nosTabs.update(tab)`

Update current js:data::*$container* tab. Can load a new URL.

Arguments

- **tab** (*mixed*) – JSON definition of tab.

  url Required. Tab URL.

  label Tab label.

labelDisplay If false, don't display the label, only the icon. Default `true`.

iconUrl Icon URL.

iconSize Icon size in pixel (square icon). Default 16.

reload If true and `url` is set, load the new URL in the current tab. Default `false`.

```
$(domContext).nosTabs('update', {
        url: 'path/url',
        label: 'Title',
        labelDisplay: true,
        iconUrl: 'path/icon.png',
        iconSize: 16
        reload: true
});
```

nosTabs('current')

`nosTabs.`**`current`**`(`*tab*`)`

Returns Index of the current js:data::*$container* tab.

```
var current = $(domContext).nosTabs('current');
```

nosToolbar

`$container.`**`nosToolbar`**`(`[*method*[, *options*[, ...]]]`)`

Manage back-office's toolbars. This is a *function with sub-functions*.

Arguments

- **method** (*mixed*) – The sub-function name, `add` (default) or `create`. If omitted, this is the first parameter of the default sub-function `nosToolbar.add()`.
- **options** (*mixed*) – Parameters of the sub-function.

nosToolbar('add')

`nosToolbar.`**`add`**`(`*element*[, *right_side*]`)`

Adds an element to the toolbar of the current js:data::*$container*. If no toolbar exists, creates a new one on-the-fly.

Arguments

- **element** (*mixed*) – Can be HTML code, a DOM element or a jQuery container.
- **right_side** (*boolean*) – Default `false`, if `true` element added at the right side of the toolbar.

```
$(domContext).nosToolbar('add', element, right_side);

\\ or
$(domContext).nosToolbar(element, right_side);

\\ Add a button, right side of toolbar
$(domContext).nosToolbar('<button>Exemple</button>', true);

\\ Add a link, left side of toolbar
var $a = $('<a href="#">Exemple</a>');
$(domContext).nosToolbar($a);
```

nosToolbar('create')

`nosToolbar.create()`

Creates a toolbar in the current js:data::*$container*.

```
$(domContext).nosToolbar('create');
```

1.2.3 Events

The back-office of Novius OS is one "big HTML page". Actions performed in one tab can affect other tabs (ex: adding, modifying or deleting an item).

An event system has been established to enable the various interface elements to communicate with each other.

On the one side, the interface elements are listening to events (by binding callbacks functions) by connecting to *dispatchers*.
On the other side, the different actions trigger events, usually returned by AJAX requests (see `$container.nosAjax()`), which are then dispatched to all interface elements via *dispatchers*.

Dispatched events are executed immediately on the current active tab or popup (the one which has focus). For other tabs (or popups), they are executed only when the tab or popup becomes active / focused.

dispatcher

A DOM element that receives system events. The child elements of the dispatcher can listen for events by connecting to it.
Each tab and popup have a dispatcher.

Structure of an event

`Event`

`Event.name`

`string`
Required
Event name. For events on a `Model`, the name is the Model name, including its PHP namespace.

`Event.id`

`int` or `[int]`
For events on a `Model`, ID(s) of the item to which they relate.

`Event.action`

`string`
Name of the action item that triggered the event. Ex: `insert`, `update` or `delete`.

`Event.context`

`string` or `[string]`

Context of the item that triggered the event. See *Multi-Contexts*.

nosListenEvent

`$container.`**`nosListenEvent`**`(`*event*, *callback*[, *caller*]`)`

Listen one (or many) event(s), i.e. register a callback function to be called when the event occurs.
Listening will be on current *dispatcher* (closest relatives in the DOM element in jQuery container).

For the callback function to be triggered, listened and triggered events should not match exactly. The listened event can just match one property of the triggered event.

Arguments

- **event** (*mixed*) – `{}` or `[{}]`. Required. JSON event to listen.
- **callback** (*function*) – Required. The callback function to execute when the event occurs. The function takes as parameter the triggered event.
- **caller** (*string*) – Caller name. If set, can stop listening to specific listener. See `$container.nosUnlistenEvent()`.

```
// Listen all events with name 'Nos\Model_Page'
$(domContext).nosListenEvent({
        name: 'Nos\Model_Page'
}, function(event) {
        // ...
}, 'caller');

// Listen all events with the 'Nos\Model_Page' name and 'insert' or 'delete' actions
$(domContext).nosListenEvent({
                name: 'Nos\Model_Page',
                action: ['insert', 'delete']
        },
        function(event) {
                // ...
        });

// Listen all events with the 'Nos\Model_Page' name and 'insert' or 'delete' actions,
// or events with the 'Nos\Model_Page' name and the 'main::en_GB' context
$(domContext).nosListenEvent([
        {
                name: 'Nos\Model_Page',
                action: ['insert', 'delete']
        },
        {
                name; 'Nos\Model_Page',
                context; 'main::en_GB'
        }
], function(event) {
        // ...
});
```

nosUnlistenEvent

`$container.`**`nosUnlistenEvent`**`(`*caller*`)`

Stop listening events for a specific caller. See `caller param of nosListenEvent`.

Arguments

- **caller** (*string*) – Caller name.

```
$(domContext).nosUnlistenEvent('caller');
```

nosDispatchEvent

`$.nosDispatchEvent(event)`

Dispatches an event to all available *dispatchers*.

Arguments

- **event** (*JSON*) – See `Event`.

```
// Dispatch event, page with ID 4 has been create with 'main::en_GB' context
$.nosDispatchEvent({
        name: 'Nos\Model_Page',
        action: 'insert',
        id: 4,
        context: 'main::en_GB',
});
```

PHP Namespace Index