Admin Documentation

Release 1.0

Marc Neuhaus

May 04, 2014

This Packages tries to provide a Base to have basic CRUD Operations with a good User Interface and Experience with the least effort and code as Possible, because i believe, that, the DRY principle should be Used for the GUI of your application as much as Possible as Well. In Principle you only need to specify one Tag called @AdminAnnotationsActive in your Model's class and this Admin package will take care of the rest. Every other Option is optional to optimize the Experience of the GUI. For Maximum Flexibility and Versatility a System of Fallback Mechanisms is in place to override the Default Template.

CHAPTER 1

General

1.1 Basic Usage

1.1.1 Installation

If you don't have a FLOW3 Project set up yet take a look at this: http://flow3.typo3.org/documentation/quickstart.html

Installing Admin:

```
cd %FLOW3-Project-Directory%
git clone git@github.com:mneuhaus/FLOW3-Admin.git Packages/Application/Admin
./flow3 package:activate Admin
./flow3 doctrine:migrate
```

Adding AdminDemo as well:

```
cd %FLOW3-Project-Directory%
git clone git@github.com:mneuhaus/FLOW3-AdminDemo.git Packages/Application/AdminDemo
./flow3 package:activate AdminDemo
./flow3 doctrine:migrate
```

1.1.2 Quick start

There are 2 Ways to Configure the Admin Interface:

1. Settings.yaml
2. Class Reflections inside the Models

> **Note:** The Settings.yaml overrules the Class Reflections in order to make it Possible to change the Behaviour of 3rd Party Packages without messing with external Code.

Settings.yaml:

```
Admin:
    Beings:
        \TYPO3\Blog\Domain\Model\Post:
            Active: true
            Properties:
                content:
                    Widget: TextArea
```

This Example Activates the Post model of the Blog Example (autoadmin:true) and Changes the Widget for the Content Property from a simple Textfield to a Textarea

Class Reflections:

```
use Admin\Annotations as Admin;
/**
 * A blog post
 * ...
 * @Admin\Active
 */
class Post {
    /**
     * @var string
     * @Admin\Widget("TextArea")
     */
    protected $content;
}
```

This Example Does the exact same thing as the Settings.yaml Example but this time inside the Post.php file with the Tag @AdminActive and @AdminWidget("TextArea")

1.2 Additional Features

1.2.1 MagicModel

You can extend your Models from this Model to get Magic Getters, Setters and some other features. Be aware of the fact that you should implement your Getters and Setters sooner or later to gain some Performance. But for development stage it just keeps the FLOW when you don't need to bother about all those repetative getters and setters all the time. > Note: This Administration interface works completely without this MagicModel. You just need to make sure, that you have all the getter and setter functions properly defined in your models. Additionally it is strongly suggested to implement the __toString funtion for your Models to return a sensible String representation of the Model.

getPropertyName() tries to get the property

setPropertyName($value) tries to set the property

addPropertyName($item) add an item to an collection

hasPropertyName($item) checks if the collection contains that item

removePropertyName($item) removes the item from the collection

__toString() returns an smart string representation of the Model

toArray() dumps the models properties to an array

fromArray($values) sets the models properties based on the supplied values

1.2.2 Navigation

Adding by Annotation

Through the NavigationAnnotation you have the ability to add any number of ControllerActions to the global Admin Navigations

title specifies the Title for the NavigationItem

position specifies the Position where this NavigationItem should be shown (top, left)

priority specify an integer of the NavigationItem's priority, NavigationItems are sorted from highest to lowest

Example:

```
use Admin\Annotations as Admin;
class StandardController extends \TYPO3\FLOW3\MVC\Controller\ActionController {
    /**
     * @return void
     * @Admin\Navigation(title="Overview", position="top", priority="10000")
     */
    public function indexAction() {
    }
}
```

Adding by API

Additionally to the Annotation method you can add items to the Navigation through the AdminCore-API::addNavigationItem($name, $position, $arguments, $priority).

name specifies the Title for the NavigationItem

position specifies the Position where this NavigationItem should be shown (top, left)

arguments arguments for the link to be generated

priority specify an integer of the NavigationItem's priority, NavigationItems are sorted from highest to lowest

> Note: Be aware of the fact, that NavigationItems added through this API aren't persisted and should only be used for the sidebar

***Example**:

```
$arguments = array(
    "action" => "index",
    "controller" => "standard",
    "package" => "AdminDemo"
);
\Admin\Core\API::addNavigationItem("MySidebarNavigationItem", "left", $arguments, 10);
```

1.2.3 Access Control

Through the Access annotation you have the ability to protect your ControllerActions with the Admin UserAuthorization.

All you need to do is to add this Annotation to the Actions you wish to protect:

```
/**
 * @Admin\Annotations\Access()
 */
public function indexAction(){}
```

When you don't specifiy any parameters it will just check for a valid user and redirect to the login it no user is logged in.

Parameters

admin set this to true in order to require an admin for this action

role set this to a specific role to require the user to be in this role. (Admin overrules this!)

CHAPTER 2

Configuration

2.1 Model Configuration

2.1.1 Active

Enable the Admin Interface for a Model

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  */
class Post {...
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Active: true
```

2.1.2 Group

Specifiy a Group in which the Model will be Listed in the Menu. By Default the Models will be Sorted in Categories based on the Package name.

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  * @Admin\Group("MyBlog")
  */
class Post {...
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Active: true
    Group: MyBlog
```

2.1.3 Label

Specifiy a Label for the Model to be used in the Menu.

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  * @Admin\Label("Blog Posts")
  */
class Post {...
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Active: true
    Label: Blog Posts
```

2.1.4 Set

By Default all [Properties](property) will be in a General Fieldset called General in the Order in which they are listed in the Models class. You can override this by specifiying specific Sets of fields.

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  * @Admin\Annotations\Set(title="Main", properties="title,content")
  * @Admin\Annotations\Set(title="Extended Informations", properties="linkTitle,date,author,image")
  */
class Post {...
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Active: true
    Set:
        -
            Title: Main
            Properties: title, content
        -
            Title: Extended Informations
            Properties: linkTitle, date, author, image
```

2.1.5 Variant

Variants are different Templates for actions. There are 3 Variants for the List Action included:

List The regular Pagniated Table

Panes Variant with 2 Panes like a E-Mail View

Calendar Very basic implementation for a calendar view

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  * @Admin\Variant(variant="Calendar", options="Calendar, List")
  */
class Event {...
```

YAML:

```
Admin\Domain\Model\Event:
    Active: true
    Variant:
        variant: Calendar
        options: Calendar, List
```

Options

variant Name of the Variant

options List of Variants that should be selectable

2.1.6 VariantMappings

VariantMappings are used in conjunction with Variants to tell the specific variant which property of the entity can be used for what

Panes image, title, subtitle, content

Calendar title, start, end

Class Reflection:

```
use Admin\Annotations as Admin;
/** A Blog post
  * ...
  * @Admin\Active
  * @Admin\Variant(variant="Calendar", options="Calendar, List")
  * @Admin\VariantMapping(title="title", start="startdate", end="enddate")
  */
class Event {...
```

YAML:

```
Admin\Domain\Model\Event:
    Active: true
    Variant:
        variant: Calendar
        options: Calendar, List
    VariantMapping:
        title: title
        start: startdate
        end: enddate
```

2.2 Property Configurations

2.2.1 Filter

By Tagging a Property with this Tag the Admin Interface will try to provide a Selectbox to Filter the List View by the Possible Values of this Property

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Filter
 */
protected $author;
```

YAML:

```
TYPO3\Blog\Domain\Model\Blog:
    Properties:
        Author:
            Filter: true
```

2.2.2 Label

With this Tag you can set the Label for a Property which is by Default a CamelCased version of the propertyname

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Label("Post Title")
 */
 protected $title;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Title:
            Label: Post Title
```

2.2.3 Ignore

There are a number of Properties which have no use to be Administrated through a GUI. With the ignore Tag you can control the Visibility of the Property to the Admin Interface.

Ignore the Property Completly

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Ignore
 */
protected $id;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Id:
            Ignore: true
```

Ignore the Property in specific Views

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Ignore list,view */
protected $content;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Content:
            Ignore: list,view
```

2.2.4 Infotext

For more Information about a Property aside from the Title you can provide an Infotext that will be shown beside/below the Input Widgets in the Create and Update Views

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\InfoText("Please tell us who you are")
 */
protexted $author;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Author:
            Infotext: Please tell us who you are
```

2.2.5 Inline

Through this annotation you can make the Child Entities of relations editable directly on the editing page of the parent entity There are 2 different variants included for this: Default and Tabular. See Variants for more informations. Make sure to add "cascade={"all"}" to your ORM relation, because otherwise you'll get an error when trying to save

Class Reflection:

```
/**
 * @var \AdminDemo\Domain\Model\Address
 * @ORM\ManyToOne(cascade={"all"})
 * @Admin\Inline()
```

```
 */
protected $address;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Address:
            Inline: True
```

2.2.6 OptionsProvider

Class Reflection:

```
/**
 * @var \Doctrine\Common\Collections\ArrayCollection<\Admin\Security\Policy>
 * @Admin\Annotations\OptionsProvider("\Admin\OptionsProvider\PolicyOptionsProvider")
 */
protected $grant;
```

YAML:

```
TYPO3\Blog\Domain\Model\Blog:
    Properties:
        Grant:
            OptionsProvider: \Admin\OptionsProvider\PolicyOptionsProvider
```

2.2.7 Representation

Through this option you can set options for the Representation of an property. Currently it is only used for the datetimeFormat

Class Reflection:

```
/**
 * @var \Datetime
  * @Admin\Representation(datetimeFormat="Y-m-d")
 */
protected $date;
```

YAML:

```
Admin\Domain\Model\Widgets:
    Properties:
        date:
            Repesentation:
                datetimeFormat: Y-m-d
```

2.2.8 Title

This option only works if you use the MagicModel! You can Specify any Property that can be Converted to a String as a Title to be used as a simple String Repesentation which is for example used in the Single and Multiple Relation Widget to Identify an Model Item

The MagicModel will try the following things to determine a title:

1. Does the Model Provide a __toString() Method
2. Does one or more @title Tags exist
3. Are there Properties Tagged as @identity which can be converted to String
4. Is there a Property with the name "title" or "name"

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Title
 */
protected $title;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Title:
            Title: true
```

First thing that matches will be used in the Order specified

2.2.9 Validate

Please Check the FLOW3 Documentation for the Validation rule: http://flow3.typo3.org/documentation/guide/partii/validation.html

2.2.10 Variant

Variants are different Variations for a similar use-case. Included are Variants for the InlineEditing:

Default The default Stacked Form-View

Tabular In this variant the inputs are aligned in a table to take up less space.

Class Reflection:

```
/**
 * @var \AdminDemo\Domain\Model\Address
 * @ORM\ManyToOne(cascade={"all"})
 * @Admin\Inline()
 * @Admin\Variant("Tabular")
 */
protected $address;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Address:
            Inline: True
            Variant: Tabular
```

2.2.11 Widget

Instead of the automatically Assigned Widget you can use this Tag to specify a specific Widget.

Class Reflection:

```
/**
 * @var string
 * @Admin\Annotations\Widget("TextArea")
 */
 protected $content;
```

YAML:

```
TYPO3\Blog\Domain\Model\Post:
    Properties:
        Content:
            Widget: TextArea
```

2.3 Templates

Since the Admin Interface would be not of much use if you could only use the Templates that the Admin Package ships with by default there is a Fallback System in place to automatically choose the most specific Template possible to Render the Admin Interface. The Fallbacks are Configured in the Settings.yaml and can be customized if needed.

2.3.1 Views

The Admin Interface will check for the existence of each of this Fallbacks until a Template is found and then Render accordingly:

```
resource://@package/Private/Templates/@being/@action/@variant.html
resource://@package/Private/Templates/Admin/@action/@variant.html
resource://@package/Private/Templates/@being/@action.html
resource://@package/Private/Templates/Admin/@action.html
resource://Admin/Private/Templates/Standard/@action/@variant.html
resource://Admin/Private/Templates/Standard/@action.html
```

@package Name of the Package which Contains the Model to be Rendered

@being Short name of the Model (TYPO3BlogDomainModelPost -> Post)

@action Action to Render (List, Create, Confirm, View, ...)

@variant Variant to Render (Tabular, Block)

2.3.2 Partials

Partials are Subparts which can be Reused in more than one View (Form, Table, Toolbar,...):

```
resource://@package/Private/Partials/@being/@action/@partial/@variant.html
resource://@package/Private/Partials/@being/@action/@partial.html
resource://@package/Private/Partials/@being/@partial/@variant.html
resource://@package/Private/Partials/@being/@partial.html
resource://@package/Private/Partials/@action/@partial/@variant.html
resource://@package/Private/Partials/@action/@partial.html
resource://@package/Private/Partials/@partial/@variant.html
resource://@package/Private/Partials/@partial.html
resource://Admin/Private/Partials/@action/@partial.html
resource://Admin/Private/Partials/@action/@partial/@variant.html
```

```
resource://Admin/Private/Partials/@partial/@variant.html
resource://Admin/Private/Partials/@partial.html
resource://Admin/Private/Partials/@partial/Default.html
```

@package Name of the Package which Contains the Model to be Rendered

@being Short name of the Model (TYPO3BlogDomainModelPost -> Post)

@partial Name of the Partial (Form, Table, Toolbar,...)

2.3.3 Widgets

```
resource://@package/Private/Partials/@being/Widgets/@partial.html
resource://@package/Private/Partials/Widgets/@partial.html
resource://Admin/Private/Partials/Widgets/@partial.html
```

@package Name of the Package which Contains the Model to be Rendered

@being Short name of the Model (TYPO3BlogDomainModelPost -> Post)

@partial Name of the Partial (TextField, Boolean, DateTime,...)

2.3.4 DashboardWidgets

```
resource://@package/Private/Partials/@being/DashboardWidgets/@partial.html
resource://@package/Private/Partials/DashboardWidgets/@partial.html
resource://Admin/Private/Partials/DashboardWidgets/@partial.html
```

@package Name of the Package which Contains the Model to be Rendered

@being Short name of the Model (TYPO3BlogDomainModelPost -> Post)

@partial Name of the Partial (TextField, Boolean, DateTime,...)

CHAPTER 3

Extending

3.1 Actions

Actions for the Admin need to implement the following interaface:

```
namespace Admin\Core\Actions;
interface ActionInterface {

        /**
         * Function to Check if this Requested Action is supported
         * @author Marc Neuhaus <mneuhaus@famelo.com>
         * */
        public function canHandle($being, $action = null, $id = false);

        /**
         * The Name of this Action
         * @author Marc Neuhaus <mneuhaus@famelo.com>
         * */
        public function __toString();

        /**
         * @param string $being
         * @param array $ids
         * @author Marc Neuhaus <mneuhaus@famelo.com>
         * */
        public function execute($being, $ids = null);

}
```

Description of the functions

canHandle(*$being*, *$action = null*, *$id = false*)
This function receives 3 arguments, based on which you need to decide if this action can ha

Parameters

- **$being** – represents the current class
- **$action** – name of current action (list, view, create, update, bulk,...)
- **$id** – specifies if this action will receive ids as well

__toString()
This functions returns a Name for this action that will be used for the Buttons and such

execute(*$being*, *$ids = null*)
This function handles the execution of the action.

Parameters

- **$being** – represents the current class
- **$ids** – an array of ids to act upon

3.1.1 Examples

The Delete action needs $ids to delete, so it returns true if there are ids to receive:

```
class DeleteAction extends \Admin\Core\Actions\AbstractAction {
    public function canHandle($being, $action = null, $id = false) {
        return $id;
    }
}
```

The Update action needs $ids to update, but can't handle bulk actions:

```
class UpdateAction extends \Admin\Core\Actions\AbstractAction {
    public function canHandle($being, $action = null, $id = false) {
        switch($action) {
            case "bulk":
                return false;
            default:
                return $id;
        }
    }
}
```

3.1.2 Rendering a view for the action

The function execute behaves exactly like a regular controllerAction. The following variables are defined in the ActionClass:

$this->request the regular controllerRequest

$this->view the view to be rendered

$this->adapter the current adapter to handle objects

$this->controller the responsible controller

3.2 Widgets

3.2.1 Available Properties in a Widget Template

Inside the Widget Partial there is one essential Variable available called {property}. This Variable Provides the following values:

property.adapter Classname of the current Adapter

property.widget Name of the Widget

property.value Unprocessed value of the Property, this might be almost anything depending on the Data in the Object. Handle this with care, because it might cause Rendering errors. In most cases you should simply use the {property.string} option to get an String representation of the Value.

property.infotext Informational Text for the Property

property.label Label for the Property

property.string String representation for the property's value

property.inputName Appropriate name for an input including the proppert prefix (item[propertyname])

property.type DataType of the property

property.name Name of the Property

3.2.2 Replace the default Widget for a specific datatype

Widgets are assigned to datatypes by a fallback system configured in Settings.yaml:

```
Doctrine:
    Widgets:
        Mapping:
            string:   Textfield
            readonly:   TextfieldReadonly
            integer:  Spinner
            float:    Textfield
            boolean:  Boolean
            \TYPO3\FLOW3\Resource\Resource: Upload
            \DateTime: DateTime
            ^\[A-Za-z]+\Domain\Model\[A-Za-z]+: SingleRelation
            ^\[A-Za-z]+\Security\[A-Za-z]+: SingleRelation
            ^\Doctrine\Common\Collections\Collection\<\*\[A-Za-z]+\Domain\Model\[A-za-z]+>: MultipleR
            ^\Doctrine\Common\Collections\Collection\<\*\[A-Za-z]+\Security\[A-za-z]+>: MultipleRelat
```

On The Left side you have your Classes or Names of the DataTypes and on the Right Side is the repsonsible Widget to use. You can override any of these Widgets in your Production/Development Settings.yaml. Aside from Classes or DataType Names you can specify an Regular Expression to Match more Complex things, like in this Case Entity Models.

3.3 Dashboard Widgets

DashboardWidgets are shown on the Admin main page and need to implement the function "initializeWidget".

3.3.1 Example

LogWidget.php:

```
/**
 *
 * @license http://www.gnu.org/licenses/lgpl.html GNU Lesser General Public License, version 3 or lat
 */
class LogWidget extends \Admin\Core\DashboardWidgets\AbstractDashboardWidget {
    public function initializeWidget() {
        $query = $this->objectManager->get("\Admin\Domain\Repository\LogRepository")->createQuery();
        $query->setOrderings(array(
```

```
            "datetime" => 'ASC'
        ));
        $query->setLimit("10");
        $logs = $query->execute();
        $this->view->assign("logs", $logs);
    }
}
```

Private/Partials/DashboardWidgets/Log.html:

```
<h5 class="well-header">Recent Activity</h5>
<f:if condition="{logs.count} > 0" >
    <f:then>
        <table class="zebra-striped condensed-table cozy">
            <thead>
            <tr>
                <th>Type</th>
                <th>Action</th>
                <th>User</th>
            </tr>
            </thead>
            <f:for each="{logs}" as="log">
                <tr>
                    <td>{log.being}</td>
                    <td>
                        <span class="label">{log.action}</span>
                    </td>
                    <td>{log.user}</td>
                </tr>
            </f:for>
        </table>
    </f:then>
    <f:else>
        No actions yet.
    </f:else>
</f:if>
```

3.4 OptionsProviders

An Options Provider creates the List of Options for the SingleRelation and MultipleRelation Widgets. Currently there is the Default implementation which creates the Options Using the ID and String Representation of the Object and one that load Options from an simple source. But for Example the PolicyOptionsProvider ensures that there are all needed Options as Policy available when the Roles Object is loaded

3.4.1 RelationOptionsProvider

This Optionsprovider gives available options based on the entity's relation

3.4.2 ArrayOptionsProvider

This Optionsprovider gives available options based on the entity's relation

Reflection:

```
/**
 * @var string
 * @Admin\Widget("Dropdown")
 * @Admin\OptionsProvider(name="Array", property="options")
 */
protected $optionsProvider;
public $options = array(
    "Hello" => "World",
    "Hell" => "Yea"
);
```

YAML:

```
TYPO3\Party\Domain\Model\ElectronicAddress:
  Properties:
    type:
      OptionsProvider:
        name: Array
        options:
          aim: Aim
          email: Email
          gizmo: Gizmo
          icq: Icq
          jabber: Jabber
          msn: Msn
          sip: Sip
          skype: Skype
          url: Url
          yahoo: Yahoo
```

3.4.3 PolicyOptionsProvider

Similar to the RelationOptionsProvider with the difference, that it populates the Policy table with policies based on available entities and actions

3.5 ViewHelpers

3.5.1 ApiViewHelper

This ViewHelper provides access to the AdminCoreAPI:: functions.

get specifies the variable or function to trigger on the API

as specifies the variable which will contain the result

3.5.2 BeingViewHelper

This ViewHelper converts an regular Object to a so called Being with all Annotations, properties etc

class class to construct a being from

object object to construct a being from

3.5.3 DashboardWidgets

This ViewHelper renders the currently active Widgets.

3.5.4 FilterViewHelper

This ViewHelper can be used to filter objects

objects the objects that should be sorted

as variable for the filtered objects. By Default: filteredObjects

filtersAs variable for the filters. By Default: filters

Example:

```
<a:query.filter objects="{ objects}">
    <f:for each="{ filteredObjects}" as="object">
        ...
    </f:for>
    <a:render partial="Filters/Right" fallbacks="Partials"/>
</a:query.paginate>
```

3.5.5 Form.FieldViewHelper

This ViewHelper renders a form field with error handling, label infotext, etc

property the beings property to render

Example:

```
<f:form method="post" action="form" fieldNamePrefix="form">
    <a:being className="AdminDemo\Domain\Model\Address" as="being">
        <a:form.field being="{ being.street}" />
        <a:form.field being="{ being.housenumber}" />
        <a:form.field being="{ being.city}" />
    </a:being>
</f:form>
```

3.5.6 Form.FieldsViewHelper

This ViewHelper renders a form for a being. This ViewHelper doesn't render the form tag itself!

being being to render the form for

Example:

```
<f:form method="post" action="form" fieldNamePrefix="form">
    <a:being className="AdminDemo\Domain\Model\Address" as="being">
        <a:form.fields being="{ being}" />
    </a:being>
</f:form>
```

3.5.7 LayoutViewHelper

This ViewHelper extends the regular LayoutViewHelper with the ability to specifiy an package to search for the layout

name name of the layout

package name of the package to look for the layout

Example:

```
<a:layout name="Bootstrap" package="Admin"/>
```

3.5.8 NavigationViewHelper

Helps by rendering previously registered Navigation Items

position specifies the region of this navigation (top, left, ...)

as specifies the variable which will contain the navigation items

Example:

```
<ul>
<a:navigation position="top" as="navItem">
    <li><a href="{ navItem.link}">{ navItem.name}</a></li>
</a:navigation>
</ul>
```

3.5.9 PaginationViewHelper

This is a simple pagination ViewHelper to limit and paginate objects

objects the objects that should be paginated

as variable for the paginated objects. By Default: paginatedObjects

limitsAs variable for the limits. By Default: limits

paginationAs variable for the pagination. By Default: pagination

Example:

```
<a:query.paginate objects="{ objects}">
    <f:for each="{ paginatedObjects}" as="object">
        ...
    </f:for>

    <div class="pagination pull-left">
        <a:render partial="Limits" fallbacks="Partials"/>
    </div>

    <div class="pagination pull-right">
        <a:render partial="Pagination" fallbacks="Partials"/>
    </div>
</a:query.paginate>
```

3.5.10 RenderViewHelper

This ViewHelper extends the regular RenderViewHelper with these features:

optional you can set the optional parameter to true in conjunction with the section attribute. In contrast to the regular RenderViewHelper this one renders it's childs if the section isn't overidden instead of an empty string

fallbacks with this function you can specify an fallback path from the settings to search for the partial in conjunction with the vars parameter

Examples(Partial):

```
<a:render partial="Pagination" fallbacks="Partials"/>
```

Examples(Section):

```
<a:render section='container' optional="true">
    Content to be rendered when this section isn't overidden
</a:render>
```

3.5.11 SettingsViewHelper

This ViewHelper gives you access to global Settings from the view

path specifies the path to the setting

3.5.12 SortViewHelper

This ViewHelper can be used to sort objects

objects the objects that should be sorted

as variable for the sorted objects. By Default: sortedObjects

sortingAs variable for the sorting. By Default: sorting

Example:

```
<a:query.sort objects="{ objects}">
    <f:link.action addQueryString="true" arguments="{sort: 'title', direction: sorting.oppositeDirect
        Sort by title
    </f:link.action>
    <f:for each="{ sortedObjects}" as="object">
        ...
    </f:for>
</a:query.paginate>
```

3.5.13 UserViewHelper

This ViewHelper gives you access to the current user

as specifies the variable which will contain the user