Neos CMS

Release 8.3.x

Neos Team and Contributors

Mar 10, 2024

CONTENTS

1 References **3**
- 1.1 NodeType Definition Reference . . . 3
- 1.2 Property Editor Reference . . . 11
- 1.3 Inspector Views Reference . . . 27
- 1.4 View Helper Reference . . . 35
- 1.5 Fusion Reference . . . 131
- 1.6 Eel Helpers Reference . . . 167
- 1.7 FlowQuery Operation Reference . . . 210
- 1.8 Neos Command Reference . . . 224
- 1.9 Validator Reference . . . 267
- 1.10 Signal Reference . . . 279
- 1.11 Coding Guideline Reference . . . 305
- 1.12 Configuration Reference . . . 322
- 1.13 Node Migration Reference . . . 323

2 Contribute **331**
- 2.1 UI Development . . . 331
- 2.2 API Documentation . . . 333

3 Neos Operations **343**
- 3.1 Command Line Tools . . . 343

4 Appendixes **347**

5 Indices and tables **349**

Neos is a free enterprise web content management system licensed under the GPL.

Our documentation can be found at https://docs.neos.io

This version of the reference documentation covering Neos 8.3.x has been rendered at: Mar 10, 2024

CHAPTER

ONE

REFERENCES

Mostly autogenerated documentation for ViewHelpers, EelHelpers, Fusion etc. from all Packages that are in a default (Demo Package) setup.

1.1 NodeType Definition Reference

THe manual to understand NodeType definitions can be found in the Neos Docs (https://docs.neos.io/cms/manual/content-repository/nodetype-definition).

The following options are allowed for defining a NodeType:

abstract
: A boolean flag, marking a node type as *abstract*. Abstract node types can never be used standalone, they will never be offered for insertion to the user in the UI, for example.

  Abstract node types are useful when using inheritance and composition, so mark base node types and mixins as abstract.

aggregate
: A boolean flag, marking a node type as *aggregate*. If a node type is marked as aggregate, it means that:

  - the node type can "live on its own", i.e. can be part of an external URL
  - when moving this node, all node variants are also moved (across all dimensions)
  - Recursive copying only happens *inside* this aggregate, and stops at nested aggregates.

  The most prominent *aggregate* is *Neos.Neos:Document* and everything which inherits from it, like *Neos.NodeTypes:Page*.

superTypes
: An array of parent node types as keys with a boolean value:

```
'Neos.Neos:Document':
  superTypes:
    'Acme.Demo.ExtraMixin': true

'Neos.Neos:Shortcut':
  superTypes:
    'Acme.Demo.ExtraMixin': false
```

constraints
: Constraint definitions stating which nested child node types are allowed. Also see the dedicated chapter Node-Type Constraints for detailed explanation:

```
constraints:
  nodeTypes:
    # ALLOW text, DISALLOW Image
    'Neos.NodeTypes:Text': true
    'Neos.NodeTypes:Image': false
    # DISALLOW as Fallback (for not-explicitly-listed node types)
    '*': false
```

childNodes

A list of child nodes that are automatically created if a node of this type is created. For each child the `type` has to be given. Additionally, for each of these child nodes, the `constraints` can be specified to override the "global" constraints per type. Here is an example:

```
childNodes:
  someChild:
    type: 'Neos.Neos:ContentCollection'
    constraints:
      nodeTypes:
        # only allow images in this ContentCollection
        'Neos.NodeTypes:Image': true
        '*': false
```

By using `position`, it is possible to define the order in which child nodes appear in the structure tree. An example may look like:

```
'Neos.NodeTypes:Page':
  childNodes:
    'someChild':
      type: 'Neos.Neos:ContentCollection'
      position: 'before main'
```

This adds a new ContentCollection called someChild to the default page. It will be positioned before the main ContentCollection that the default page has. The position setting follows the same sorting logic used in Fusion (see the *Fusion Reference*).

label

When displaying a node inside the Neos UI (e.g. tree view, link editor, workspace module) the `label` option will be used to generate a human readable text for a specific node instance (in contrast to the `ui.label` which is used for all nodes of that type).

The label option accepts an Eel expression that has access to the current node using the `node` context variable. It is recommended to customize the *label* option for node types that do not yield a sufficient description using the default configuration.

Example:

```
'Neos.Demo:Flickr':
  label: ${'Flickr plugin (' + q(node).property('tags') + ')'}
```

generatorClass

Alternatively the class of a node label generator implementing `Neos\ContentRepository\Domain\Model\NodeLabelGeneratorInterface` can be specified as a nested option.

options

Options for third party-code, the Content-Repository ignores those options but Neos or Packages may use this to adjust their behavior.

fusion
Options to control the behavior of fusion-for a specific nodeType.

prototypeGenerator
The class that is used to generate the default fusion-prototype for this nodeType.

If this option is set to a className the class has to implement the interface `\Neos\Neos\Domain\Service\DefaultPrototypeGeneratorInterface` and is used to generate the prototype-code for this node.

If `options.fusion.prototypeGenerator` is set to `null` no prototype is created for this type.

By default Neos has generators for all nodes of type `Neos.Neos:Node` and creates protoypes based on `Neos.Fusion:Template`. A template path is assumed based on the package-prefix and the nodetype-name. All properties of the node are passed to the template. For the nodeTypes of type `Neos.Neos:Document`, `Neos.Neos:Content` and `Neos.Neos:Plugin` the corresponding prototype is used as base-prototype.

Example:

```
prototype(Vendor.Site:Content.SpecialNodeType) < prototype(Neos.
↪Fusion:Content) {
  templatePath = 'resource://Vendor.Site/Private/Templates/NodeTypes/
↪Content.SpecialNodeType.html'
  # all properties of the nodeType are passed to the template
  date = ${q(node).property('date')}
  # inline-editable strings additionally get the convertUris processor
  title = ${q(node).property('title')}
  title.@process.convertUris = Neos.Neos:ConvertUris
}
```

ui
Configuration options related to the user interface representation of the node type

label
The human-readable label of the node type

group
Name of the group this content element is grouped into for the 'New Content Element' dialog. It can only be created through the user interface if `group` is defined and it is valid.

All valid groups are given in the `Neos.Neos.nodeTypes.groups` setting

position
Position inside the group this content element is grouped into for the 'New Content Element' dialog. Small numbers are sorted on top.

icon
This setting defines the icon that the Neos UI will use to display the node type. The icon can contain a custom SVG icon as a resource URI or a Fontawesome icon.

Resource icon: For custom SVG icons a resource URI to the file needs to be configured. e.g. 'resource://Neos.Demo/Images/logo.svg'

Fontawesome icon: All free Fontawesome 5 icons can be used: https://fontawesome.com/v5/search?o=r&m=free

Those can be referenced via "icon-[name]", as the UI includes a fallback to the "fas" prefix-classes. To be sure which icon will be used, they can also be referenced by their icon-classes, e.g. "fas fa-check".

`previewIcon` This setting defines the icon that will be used in the tile view of the NodeType selection dialog. It is an additional icon that can be used to have more detailed icons for the node type. So that the users get a better idea of what the node type is for.

Preview icons are by default bit bigger. We scale them to the doubled size. It is also possible to adjust the scaling by the previewIconSize setting. It can be a Fontawesome icon name or an SVG Asset Resource URL like the icon.

`previewIconSize` The previewIconSize setting defines the size of the previewIcon. By default, the previewIcon has the size "2x".

The following options are available: - 'xs' - 'sm' - 'lg' - '2x' - '3x'

`help`

Configuration of contextual help. Displays a message that is rendered as popover when the user clicks the help icon in an insert node dialog.

`message`

Help text for the node type. It supports markdown to format the help text and can be translated (see NodeType Translations).

`thumbnail`

This is shown in the popover and can be supplied in two ways:

- as an absolute URL to an image (`http://static/acme.com/thumbnails/bar.png`)
- as a resource URI (`resource://AcmeCom.Website/NodeTypes/Thumbnails/foo.png`)

If the `thumbnail` setting is undefined but an image matching the nodetype name

is found, it will be used automatically. It will be looked for in `<packageKey>/Resources/Public/NodeTypes/Thumbnails/<nodeTypeName>.png` with `packageKey` and `nodeTypeName` being extracted from the full nodetype name like this:

`AcmeCom.Website:FooWithBar` -> `AcmeCom.Website` and `FooWithBar`

The image will be downscaled to a width of 342 pixels, so it should either be that size to be placed above any further help text (if supplied) or be half that size for the help text to flow around it.

`inlineEditable`

If *true*, it is possible to interact with this Node directly in the content view. If *false*, an overlay is shown preventing any interaction with the node. If not given, checks if any property is marked as `ui.inlineEditable`.

`inspector`

These settings configure the inspector in the Neos UI for the node type

`tabs`

Defines an inspector tab that can be used to group property groups of the node type

`label`

The human-readable label for this inspector tab

`position`

Position of the inspector tab, small numbers are sorted on top

`icon`

This setting define the icon to use in the Neos UI for the tab

Currently it's only possible to use a predefined selection of icons, which are available in Font Awesome http://fortawesome.github.io/Font-Awesome/3.2.1/icons/.

`groups`

Defines an inspector group that can be used to group properties of the node type

`label`
The human-readable label for this inspector group

`position`
Position of the inspector group, small numbers are sorted on top

`icon`
This setting define the icon to use in the Neos UI for the group

`tab`
The tab the group belongs to. If left empty the group is added to the `default` tab.

`collapsed`
If the group should be collapsed by default (true or false). If left empty, the group will be expanded.

`views`
Defines views that can be used to display read-only data alongside property editors inside the inspector

`label`
The human-readable label for this view

`group`
Identifier of the *inspector group* this view is categorized into in the content editing user interface. If none is given, the view is not visibile in the property inspector of the user interface.

The value here must reference a group configured in the `ui.inspector.groups` element of the node type this view belongs to.

`position`
Position inside the inspector group, small numbers are sorted on top.

`view`
Name of the JavaScript View Class which is instantiated to edit this element in the inspector.

`viewOptions`
A set of options for the given view, see the *Inspector Views Reference*.

`creationDialog`
Creation dialog elements configuration. See Node Creation Dialog Configuration for more details.

`properties`
A list of named properties for this node type. For each property the following settings are available.

Note: Your own property names should never start with an underscore _ as that is used for internal properties or as an internal prefix.

`type`
Data type of this property. This may be a simple type (like in PHP), a fully qualified PHP class name, or one of these three special types: `DateTime`, `references`, or `reference`. Use `DateTime` to store dates / time as a DateTime object. Use `reference` and `references` to store references that point to other nodes. `reference` only accepts a single node or node identifier, while `references` accepts an array of nodes or node identifiers.

`defaultValue`
Default value of this property. Used at node creation time. Type must match specified 'type'.

`ui`
Configuration options related to the user interface representation of the property

`label`
The human-readable label of the property

help
Configuration of contextual help. Displays a message that is rendered as popover when the user clicks the help icon in the inspector.

message
Help text for this property. It supports markdown to format the help text and can be translated (see NodeType Translations).

reloadIfChanged
If *true*, the whole content element needs to be re-rendered on the server side if the value changes. This only works for properties which are displayed inside the property inspector, i.e. for properties which have a `group` set.

reloadPageIfChanged
If *true*, the whole page needs to be re-rendered on the server side if the value changes. This only works for properties which are displayed inside the property inspector, i.e. for properties which have a `group` set.

inlineEditable
If *true*, this property is inline editable, i.e. edited directly on the page.

`inline`

editor
The default inline editor is the CKEditor5.

editorOptions
This section controls the text formatting options the user has available for this property.

placeholder
A text that is shown when the field is empty. Supports i18n.

autoparagraph
When configured to false, automatic creation of paragraphs is disabled for this property and <enter> key would create soft line breaks instead (equivalent to configuring an editable on a span tag).

linking
A way to configure additional options available for a link, e.g. target or rel attributes.

formatting
Various formatting options (see example below for all available options).

Example:

```
inline:
  editorOptions:
    placeholder: i18n
    autoparagraph: true
    linking:
      anchor: true
      title: true
      relNofollow: true
      targetBlank: true
    formatting:
      strong: true
      em: true
      sub: true
      sup: true
```

(continues on next page)

(continued from previous page)

```
      p: true
      h1: true
      h2: true
      h3: true
      h4: true
      h5: true
      h6: true
      pre: true
      underline: true
      strikethrough: true
      removeFormat: true
      left: true
      right: true
      center: true
      justify: true
      table: true
      ol: true
      ul: true
      a: true
```

inspector
These settings configure the inspector in the Neos UI for the property.

group
Identifier of the *inspector group* this property is categorized into in the content editing user interface. If none is given, the property is not editable through the property inspector of the user interface.

The value here must reference a groups configured in the `ui.inspector.groups` element of the node type this property belongs to.

position
Position inside the inspector group, small numbers are sorted on top.

editor
Name of the JavaScript Editor Class which is instantiated to edit this element in the inspector.

editorOptions
A set of options for the given editor, see the *Property Editor Reference*.

editorListeners (removed since Neos 3.3)
This feature has been removed in favor of Depending Properties with Neos 3.3

showInCreationDialog (since Neos 5.1)
If *true* the corresponding property will appear in the Node Creation Dialog. Editor configuration will be copied from the respective `ui.inspector` settings in that case and can be overridden with the `creationDialog.elements.<propertyName>`, see Node Creation Dialog Configuration

validation
A list of validators to use on the property. Below each validator type any options for the validator can be given. See below for more information.

Tip: Unset a property by setting the property configuration to null (~).

Here is one of the standard Neos node types (slightly shortened):

```
'Neos.NodeTypes:Image':
  superTypes:
    'Neos.Neos:Content': true
  ui:
    label: 'Image'
    icon: 'icon-picture'
    inspector:
      groups:
        image:
          label: 'Image'
          icon: 'icon-image'
          position: 5
  properties:
    image:
      type: Neos\Media\Domain\Model\ImageInterface
      ui:
        label: 'Image'
        reloadIfChanged: true
        inspector:
          group: 'image'
    alignment:
      type: string
      defaultValue: ''
      ui:
        label: 'Alignment'
        reloadIfChanged: true
        inspector:
          group: 'image'
          editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
          editorOptions:
            placeholder: 'Default'
            values:
              '':
                label: ''
              center:
                label: 'Center'
              left:
                label: 'Left'
              right:
                label: 'Right'
    alternativeText:
      type: string
      ui:
        label: 'Alternative text'
        reloadIfChanged: true
        inspector:
          group: 'image'
      validation:
        'Neos.Neos/Validation/StringLengthValidator':
          minimum: 1
          maximum: 255
    hasCaption:
      type: boolean
```

(continues on next page)

(continued from previous page)

```
    ui:
      label: 'Enable caption'
      reloadIfChanged: true
      inspector:
        group: 'image'
  caption:
    type: string
    defaultValue: '<p>Enter caption here</p>'
    ui:
      inlineEditable: true
```

1.2 Property Editor Reference

For each property which is defined in `NodeTypes.yaml`, the editor inside the Neos inspector can be customized using various options. Here follows the reference for each property type.

Note: All NodeType inspector configuration values are dynamically evaluated on the client-side, see dynamic-configuration-processing for more details.

1.2.1 Property Type: boolean `BooleanEditor` – Checkbox editor

A `boolean` value is rendered using a checkbox in the inspector:

```
'isActive':
  type: boolean
  ui:
    label: 'is active'
    inspector:
      group: 'document'
```

Options Reference:

`disabled` (boolean)
HTML `disabled` property. If `true`, disable this checkbox.

1.2.2 Property Type: string `TextFieldEditor` – Single-line Text Editor (default)

Example:

```
subtitle:
  type: string
  ui:
    label: 'Subtitle'
    help:
      message: 'Enter some help text for the editors here. The text will be shown via␣
↪click.'
    inspector:
```

(continues on next page)

(continued from previous page)

```
      group: 'document'
      editorOptions:
        placeholder: 'Enter subtitle here'
        maxlength: 20
```

Options Reference:

placeholder (string)
HTML5 `placeholder` property, which is shown if the text field is empty.

disabled (boolean)
HTML `disabled` property. If `true`, disable this textfield.

maxlength (integer)
HTML `maxlength` property. Maximum number of characters allowed to be entered.

readonly (boolean)
HTML `readonly` property. If `true`, this field is cannot be written to.

form (optional)
HTML5 `form` property.

selectionDirection (optional)
HTML5 `selectionDirection` property.

spellcheck (optional)
HTML5 `spellcheck` property.

required (boolean)
HTML5 `required` property. If `true`, input is required.

title (boolean)
HTML `title` property.

autocapitalize (boolean)
Custom HTML `autocapitalize` property.

autocorrect (boolean)
Custom HTML `autocorrect` property.

1.2.3 Property Type: string `TextAreaEditor` – Multi-line Text Editor

In case the text input should span multiple lines, a `TextAreaEditor` should be used as follows:

```
'description':
    type: 'string'
    ui:
      label: 'Description'
      inspector:
        group: 'document'
        editor: 'Neos.Neos/Inspector/Editors/TextAreaEditor'
        editorOptions:
          rows: 7
```

Options Reference:

rows (integer)
Number of lines this textarea should have; Default `5`.

** and all options from Text Field Editor – see above**

1.2.4 Property Type: string `RichTextEditor` – Full-Screen Rich Text Editor

In case a large block of text has to be edited with support for rich text editing, a `RichTextEditor` can be used.

It takes all the same configuration options as the inline rich text editor under `editorOptions`:

```
'source':
    type: 'string'
    ui:
      label: 'Toggle the editor'
      inspector:
        editor: 'Neos.Neos/Inspector/Editors/RichTextEditor'
        editorOptions:
          placeholder: '<p>placeholder</p>'
          autoparagraph: true
          linking:
            anchor: true
            title: true
            relNofollow: true
            targetBlank: true
            download: true
          formatting:
            strong: true
            em: true
            u: true
            sub: true
            sup: true
            del: true
            p: true
            h1: true
            h2: true
            h3: true
            h4: true
            h5: true
            h6: true
            pre: true
            underline: true
            strikethrough: true
            removeFormat: true
            left: true
            right: true
            center: true
            justify: true
            table: true
            ol: true
            ul: true
            a: true
```

1.2.5 Property Type: string `CodeEditor` – Full-Screen Code Editor

In case HTML source code or any other plain text has to be edited, a `CodeEditor` can be used:

```
'source':
    type: 'string'
    ui:
      label: 'Source'
      inspector:
        group: 'document'
        editor: 'Neos.Neos/Inspector/Editors/CodeEditor'
```

Furthermore, the button label can be adjusted by specifying `buttonLabel`. Furthermore, the highlighting mode can be customized, which is helpful for editing markdown and similar contents:

```
'markdown':
    type: 'string'
    ui:
      label: 'Markdown'
      inspector:
        group: 'document'
        editor: 'Neos.Neos/Inspector/Editors/CodeEditor'
        editorOptions:
          buttonLabel: 'Edit Markdown'
          highlightingMode: 'text/plain'
```

Options Reference:

`buttonLabel` (string)
: label of the button which is used to open the full-screen editor. Default `Edit code`.

`highlightingMode` (string)
: CodeMirror highlighting mode to use. These formats are support by default: `text/plain`, `text/xml`, `text/html`, `text/css`, `text/javascript`. If other highlighting modes shall be used, they must be loaded beforehand using custom JS code. Default `text/html`.

`disabled` (boolean)
: If `true`, disables the CodeEditor.

1.2.6 Property Type: string / array<string> `SelectBoxEditor` – Dropdown Select Editor

In case only fixed entries are allowed to be chosen a select box can be used - multiple selection is supported as well. The data for populating the select box can be fetched from a fixed set of entries defined in YAML or a datasource. The most important option is called `values`, containing the choices which can be made. If wanted, an icon can be displayed for each choice by setting the `icon` class appropriately.

Basic Example – simple select box:

```
targetMode:
  type: string
  defaultValue: 'firstChildNode'
  ui:
    label: 'Target mode'
    inspector:
```

(continues on next page)

(continued from previous page)

```
    group: 'document'
    editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
    editorOptions:
      values:
        firstChildNode:
          label: 'First child node'
          icon: 'icon-legal'
        parentNode:
          label: 'Parent node'
          icon: 'icon-fire'
        selectedTarget:
          label: 'Selected target'
```

If the selection list should be grouped, this can be done by setting the `group` key of each individual value:

```
country:
  type: string
  ui:
    label: 'Country'
    inspector:
      group: 'document'
      editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
      editorOptions:
        values:
          italy:
            label: 'Italy'
            group: 'Southern Europe'
          austria:
            label: 'Austria'
            group: 'Central Europe'
          germany:
            label: 'Germany'
            group: 'Central Europe'
```

Furthermore, multiple selection is also possible, by setting `multiple` to `true`, which is automatically set for properties of type `array`. If an empty value is allowed as well, `allowEmpty` should be set to `true` and `placeholder` should be set to a helpful text:

```
styleOptions:
  type: array
  ui:
    label: 'Styling Options'
    inspector:
      group: 'document'
      editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
      editorOptions:

        # The next line is set automatically for type array
        # multiple: true

        allowEmpty: true
        placeholder: 'Select Styling Options'
```

(continues on next page)

(continued from previous page)

```
        values:
          leftColumn:
            label: 'Show Left Column'
          rightColumn:
            label: 'Show Right Column'
```

Because selection options shall be fetched from server-side code frequently, the Select Box Editor contains support for so-called *data sources*, by setting a `dataSourceIdentifier`, or optionally a `dataSourceUri`. This helps to provide data to the editing interface without having to define routes, policies or a controller. You can provide an array of `dataSourceAdditionalData` that will be sent to the data source with each request, the key/value pairs can be accessed in the `$arguments` array passed to `getData()`.

```
questions:
  ui:
    inspector:
      editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
      editorOptions:
        dataSourceIdentifier: 'questions'
        # alternatively using a custom uri:
        # dataSourceUri: 'custom-route/end-point'
        dataSourceAdditionalData:
          apiKey: 'foo-bar-baz'
```

See data-sources for more details on implementing a *data source* based on Neos conventions. If you are using a data source to populate SelectBoxEditor instances it has to be matching the `values` option. Make sure you sort by group first, if using the grouping option.

Example for returning compatible data:

```
return array(
    array('value' => 'key', 'label' => 'Foo', 'group' => 'A', 'icon' => 'icon-key'),
    array('value' => 'fire', 'label' => 'Fire', 'group' => 'A', 'icon' => 'icon-fire'),
    array('value' => 'legal', 'label' => 'Legal', 'group' => 'B', 'icon' => 'icon-legal')
);
```

If you use the `dataSourceUri` option to connect to an arbitrary service, make sure the output of the data source is a JSON formatted array matching the following structure. Make sure you sort by group first, if using the grouping option.

Example for compatible data:

```
[{
  "value": "key",
  "label": "Key",
  "group": "A",
  "icon": "icon-key"
},
{
  "value": "fire",
  "label": "Fire",
  "group": "A",
  "icon": "icon-fire"
},
{
```

(continues on next page)

(continued from previous page)

```
  "value": "legal",
  "label": "Legal",
  "group": "B",
  "icon": "icon-legal"
}]
```

Options Reference:

values (required array)
the list of values which can be chosen from

`[valueKey]`

label (required string)
label of this value.

group (string)
group of this value.

icon (string)
CSS icon class for this value.

allowEmpty (boolean)
if true, it is allowed to choose an empty value.

placeholder (string)
placeholder text which is shown if nothing is selected. Only works if `allowEmpty` is `true`. Default `Choose`.

multiple (boolean)
If `true`, multi-selection is allowed. Default `FALSE`.

minimumResultsForSearch (integer)
The minimum amount of items in the select before showing a search box, if set to `-1` the search box will never be shown.

dataSourceUri (string)
If set, this URI will be called for loading the options of the select field.

dataSourceIdentifier (string)
If set, a server-side data source will be called for loading the possible options of the select field.

dataSourceAdditionalData (array)
Key/value pairs that will be sent to the server-side data source with every request.

disabled (boolean)
If `true`, disables the SelectBoxEditor.

1.2.7 Property Type: string `LinkEditor` – Link Editor for internal, external and asset links

If internal links to other nodes, external links or asset links shall be editable at some point, the `LinkEditor` can be used to edit a link:

```
myLink:
  type: string
  ui:
    inspector:
      editor: 'Neos.Neos/Inspector/Editors/LinkEditor'
```

The searchbox will accept:

- node document titles
- asset titles and tags
- valid URLs
- valid email addresses

By default, links to generic `Neos.Neos:Document` nodes are allowed; but by setting the `nodeTypes` option, this can be further restricted (like with the `reference` editor). Additionally, links to assets can be disabled by setting `assets` to `FALSE`. Links to external URLs are always possible. If you need a reference towards only an asset, use the `asset` property type; for a reference to another node, use the `reference` property type. Furthermore, the placeholder text can be customized by setting the `placeholder` option:

```
myExternalLink:
  type: string
  ui:
    inspector:
      group: 'document'
      editor: 'Neos.Neos/Inspector/Editors/LinkEditor'
      editorOptions:
        assets: FALSE
        nodeTypes: ['Neos.Neos:Shortcut']
        placeholder: 'Paste a link, or type to search for nodes'
```

Options Reference:

`disabled` (boolean)
If `true`, disables the LinkEditor.

1.2.8 Property Type: integer `TextFieldEditor`

Example:

```
cropAfterCharacters:
  type: integer
  ui:
    label: 'Crop after characters'
    inspector:
      group: 'document'
```

Options Reference:

all TextFieldEditor options apply

1.2.9 Property Type: string / integer `RangeEditor` – Range Editor for selecting numeric values

The minimum, maximum and step size can be defined. Additionally, a unit label as well as a special label for the minimum and maximum value can be defined.

If a certain value should be entered, the current value can also be clicked to enter the desired value directly.

```
opacity:
  type: integer
  ui:
    inspector:
      editor: 'Neos.Neos/Inspector/Editors/RangeEditor'
      editorOptions:
        minLabel: Invisible
        maxLabel: Opaque
        min: 0
        max: 100
        step: 5
        unit: px
```

Options Reference:

`min` (integer)
The lowest value in the range of permitted values. This value must be less than or equal to the value of the max attribute.

`max` (integer)
The greatest value in the range of permitted values. This value must be greater than or equal to the value of the min attribute.

`step` (integer)
The step attribute is a number that specifies the granularity that the value must adhere to.

`unit` (string)
The value gets displayed beside the current value, as well after the minimal value (only if `minLabel` is not set) and after the maximal value (only if `maxLabel` is not set). (The unit is just a visual indicator and will not be added to the resulting property value.)

`minLabel` (string)
If set, this value is displayed instead of the minimum value.

`maxLabel` (string)
If set, this value is displayed instead of the maximum value.

`disabled` (boolean)
If set to `true`, the range editor gets disabled.

1.2.10 Property Type: reference / references `ReferenceEditor` / `ReferencesEditor` – Reference Selection Editors

The most important option for the property type `reference` and `references` is `nodeTypes`, which allows to restrict the type of the target nodes which can be selected in the editor.

Example:

```
authors:
  type: references
  ui:
    label: 'Article Authors'
    inspector:
      group: 'document'
      editorOptions:
        nodeTypes: ['My.Website:Author']
```

Options Reference:

`nodeTypes` (array of strings)
List of node types which are allowed to be selected. By default, is set to `Neos.Neos:Document`, allowing only to choose other document nodes.

`placeholder` (string)
Placeholder text to be shown if nothing is selected

`startingPoint` (string)
The starting point (node path) for finding possible nodes to create a reference. This allows to search for nodes outside the current site. If not given, nodes will be searched for in the current site. For all nodes outside the current site the node path is shown instead of the url path.

`threshold` (number)
Minimum amount of characters which trigger a search. Default is set to 2.

`createNew` (array)
It is also possible to create new selectable nodes directly from the reference editor. This can come in handy for example if you reference tag nodes and want to add new tags on the fly.

The given string is passed to the title property of the new node.

`path` (string)
The path to the node in which the new nodes should be created.

`type` (string)
The type of the nodes to be created.

```
tags:
  type: references
  ui:
    label: 'Tags'
    inspector:
      group: document
      editorOptions:
        nodeTypes: ['My.Website:Tag']
        createNew:
          path: /sites/yoursite/tags
          type: 'My.Website:Tag'
```

disabled (boolean)
If `true`, disables the Reference(s)Editor.

1.2.11 Property Type: DateTime `DateTimeEditor` – Date & Time Selection Editor

The most important option for `DateTime` properties is the `format`, which is configured like in PHP, as the following examples show:

- `d-m-Y`: `05-12-2014` – allows to set only the date
- `d-m-Y H:i`: `05-12-2014 17:07` – allows to set date and time
- `H:i`: `17:07` – allows to set only the time

Example:

```
publishingDate:
  type: DateTime
  defaultValue: 'today midnight'
  ui:
    label: 'Publishing Date'
    inspector:
      group: 'document'
      position: 10
      editorOptions:
        format: 'd.m.Y'
```

Options Reference:

`format` (required string)
The date format, a combination of y, Y, F, m, M, n, t, d, D, j, l, N, S, w, a, A, g, G, h, H, i, s. Default `d-m-Y`.

`defaultValue` (string)
Sets property value, when the node is created. Accepted values are whatever `strtotime()` can parse, but it works best with relative formats like `tomorrow 09:00` etc. Use `now` to set current date and time.

`placeholder` (string)
The placeholder shown when no date is selected

`minuteStep` (integer)
The granularity on which a time can be selected. Example: If set to `30`, only half-hour increments of time can be chosen. Default `5` minutes.

For the date format, these are the available placeholders:

- **year**
  - `y`: A two digit representation of a year - Examples: 99 or 03
  - `Y`: A full numeric representation of a year, 4 digits - Examples: 1999 or 2003
- **month**
  - `F`: A full textual representation of a month, such as January or March - January through December
  - `m`: Numeric representation of a month, with leading zeros - 01 through 12
  - `M`: A short textual representation of a month, three letters - Jan through Dec
  - `n`: Numeric representation of a month, without leading zeros - 1 through 12
  - `t`: Number of days in the given month - 28 through 31

- **day**
  - `d`: Day of the month, 2 digits with leading zeros - 01 to 31
  - `D`: A textual representation of a day, three letters - Mon through Sun
  - `j`: Day of the month without leading zeros - 1 to 31
  - `l`: A full textual representation of the day of the week - Sunday through Saturday
  - `N`: ISO-8601 numeric representation of the day of the week - 1 (for Monday) through 7 (for Sunday)
  - `S`: English ordinal suffix for the day of the month, 2 characters - st, nd, rd or th.
  - `w`: Numeric representation of the day of the week - 0 (for Sunday) through 6 (for Saturday)
- **hour**
  - `a`: Lowercase Ante meridiem and Post meridiem - am or pm
  - `A`: Uppercase Ante meridiem and Post meridiem - AM or PM
  - `g`: hour without leading zeros - 12-hour format - 1 through 12
  - `G`: hour without leading zeros - 24-hour format - 0 through 23
  - `h`: 12-hour format of an hour with leading zeros - 01 through 12
  - `H`: 24-hour format of an hour with leading zeros - 00 through 23
- **minute**
  - `i`: minutes, 2 digits with leading zeros - 00 to 59
- **second**
  - `s`: seconds, 2 digits with leading zeros - 00 through 59

`disabled` (boolean)
If `true`, disables the DateTimeEditor.

1.2.12 Property Type: image (Neos\Media\Domain\Model\ImageInterface) `ImageEditor` – Image Selection/Upload Editor

For properties of type `Neos\Media\Domain\Model\ImageInterface`, an image editor is rendered. If you want cropping and resizing functionality, you need to set `features.crop` and `features.resize` to `true`, as in the following example:

```
'teaserImage':
  type: 'Neos\Media\Domain\Model\ImageInterface'
  ui:
    label: 'Teaser Image'
    inspector:
      group: 'document'
      editorOptions:
        features:
          crop: true
          resize: true
```

If cropping is enabled, you might want to enforce a certain aspect ratio, which can be done by setting `crop.aspectRatio.locked.width` and `crop.aspectRatio.locked.height`. To show the crop dialog automatically on image upload, configure the `crop.aspectRatio.forceCrop` option. In the following example, the image format must be `16:9`:

```
'teaserImage':
  type: 'Neos\Media\Domain\Model\ImageInterface'
  ui:
    label: 'Teaser Image'
    inspector:
      group: 'document'
      editorOptions:
        features:
          crop: true
        constraints:
          mediaTypes: ['image/png']
        crop:
          aspectRatio:
            forceCrop: true
            locked:
              width: 16
              height: 9
```

If not locking the cropping to a specific ratio, a set of predefined ratios can be chosen by the user. Elements can be added or removed from this list underneath `crop.aspectRatio.options`. If the aspect ratio of the original image shall be added to the list, `crop.aspectRatio.enableOriginal` must be set to `true`. If the user should be allowed to choose a custom aspect ratio, set `crop.aspectRatio.allowCustom` to `true`:

```
'teaserImage':
  type: 'Neos\Media\Domain\Model\ImageInterface'
  ui:
    label: 'Teaser Image'
    inspector:
      group: 'document'
      editorOptions:
        constraints:
          mediaTypes: ['image/png']
        features:
          crop: true
        crop:
          aspectRatio:
            options:
              square:
                width: 1
                height: 1
                label: 'Square'
              fourFive:
                width: 4
                height: 5
              # disable this ratio (if it was defined in a supertype)
              fiveSeven: ~
            enableOriginal: true
            allowCustom: true
```

Options Reference:

maximumFileSize (string)
: Set the maximum allowed file size to be uploaded. Accepts numeric or formatted string values, e.g. "204800" or "204800b" or "2kb". Defaults to the maximum allowed upload size configured in php.ini

accept (string)
DEPRECATED. Use `constraints.mediaTypes` instead

`constraints`

mediaTypes (array)
If set, the media browser and file upload will be limited to assets with the specified media type. Default `['image/*']` Example: `['image/png', 'image/jpeg']` Note: Due to technical limitations the media browser currently ignores the media sub type, so `image/png` has the same effect as `image/*`

assetSources (array)
If set, the media browser will be limited to assets of the specified asset source. Default: `[]` (all asset sources) Example: `['neos', 'custom_asset_source]`

`features`

crop (boolean)
If `true`, enable image cropping. Default `true`.

upload (boolean)
If `true`, enable Upload button, allowing new files to be uploaded directly in the editor. Default `true`.

mediaBrowser (boolean)
If `true`, enable Media Browser button. Default `true`.

resize (boolean)
If `true`, enable image resizing. Default `FALSE`.

crop
crop-related options. Only relevant if `features.crop` is enabled.

`aspectRatio`

forceCrop (boolean)
Show the crop dialog on image upload

locked
Locks the aspect ratio to a specific width/height ratio

width (integer)
width of the aspect ratio which shall be enforced

height (integer)
height of the aspect ratio which shall be enforced

options
aspect-ratio presets. Only effective if `locked` is not set.

`[presetIdentifier]`

width (required integer)
the width of the aspect ratio preset

height (required integer)
the height of the aspect ratio preset

label (string)
a human-readable name of the aspect ratio preset

enableOriginal (boolean)
If `true`, the image ratio of the original image can be chosen in the selector. Only effective if `locked` is not set. Default `true`.

allowCustom (boolean)
If `true`, a completely custom image ratio can be chosen. Only effective if `locked` is not set. Default `true`.

defaultOption (string)
default aspect ratio option to be chosen if no cropping has been applied already.

disabled (boolean)
If `true`, disables the ImageEditor.

1.2.13 Property Type: asset (Neos\Media\Domain\Model\Asset / array<Neos\Media\Domain\Model\Asset>) `AssetEditor` – File Selection Editor

If an asset, i.e. `Neos\Media\Domain\Model\Asset`, shall be uploaded or selected, the following configuration is an example:

```
'caseStudyPdf':
  type: 'Neos\Media\Domain\Model\Asset'
  ui:
    label: 'Case Study PDF'
    inspector:
      group: 'document'
```

Conversely, if multiple assets shall be uploaded, use `array<Neos\Media\Domain\Model\Asset>` as type:

```
'caseStudies':
  type: 'array<Neos\Media\Domain\Model\Asset>'
  ui:
    label: 'Case Study PDF'
    inspector:
      group: 'document'
```

Options Reference:

accept (string)
DEPRECATED. Use `constraints.mediaTypes` instead

`constraints`

mediaTypes (array)
If set, the media browser, file search and file upload will be limited to assets with the specified media type. Default `[]` (all media types) Example: `['application/msword', 'application/pdf']` Note: Due to technical limitations the media browser currently ignores the media sub type, so `application/pdf` has the same effect as `application/*`.

assetSources (array)
If set, the media browser and file search will be limited to assets of the specified asset source. Default: `[]` (all asset sources) Example: `['neos', 'custom_asset_source]`

`features`

upload (boolean)
If `true`, enable Upload button, allowing new files to be uploaded directly in the editor. Default `true`.

mediaBrowser (boolean)
If `true`, enable Media Browser button. Default `true`.

disabled (boolean)
If true, disables the AssetEditor.

Property Validation

The validators that can be assigned to properties in the node type configuration are used on properties that are edited via the inspector or inline. They are applied on the client-side only. The available validators can be found in the Neos package in Resources/Public/JavaScript/Shared/Validation:

- AlphanumericValidator
- CountValidator
- DateTimeRangeValidator
- DateTimeValidator
- EmailAddressValidator
- FloatValidator
- IntegerValidator
- LabelValidator
- NotEmptyValidator
- NumberRangeValidator
- RegularExpressionValidator
- StringLengthValidator
- StringValidator
- TextValidator
- UuidValidator

The options are in sync with the Flow validators, so feel free to check the Flow documentation for details.

To apply options, just specify them like this:

```
someProperty:
  validation:
    'Neos.Neos/Validation/StringLengthValidator':
      minimum: 1
      maximum: 255
```

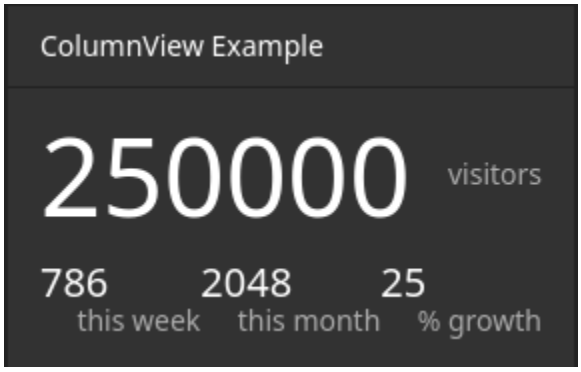
Extensibility

It is also possible to add custom-editors and use custom-validators.

1.3 Inspector Views Reference

Each `NodeTypes.yaml` can be used to configure views that are displayed inside the Neos inspector. Here follows the reference for each built-in view.

1.3.1 Data/ColumnView



The `ColumnView` displays numeric data from a data source as a compact view.

A data source is a PHP class implementing `Neos\Neos\Service\DataSource\DataSourceInterface` (which is best achieved by extending `Neos\Neos\Service\DataSource\AbstractDataSource`). The example displayed above uses the following data source implementation:

```php
<?php

/*
 * This script belongs to the package "Vendor.Site".
 */

declare(strict_types=1);

namespace Vendor\Site\Application\Neos\DataSource;

use Neos\Neos\Service\DataSource\AbstractDataSource;
use Neos\ContentRepository\Domain\Model\NodeInterface;

final class ColumnViewDataSource extends AbstractDataSource
{
    /**
     * @var string
     */
    static protected $identifier = 'vendor-site-column-view';

    /**
     * Get data
     *
     * {@inheritdoc}
     */
    public function getData(NodeInterface $node = NULL, array $arguments = [])
    {
```

(continues on next page)

(continued from previous page)

```
        return [
            'data' => [
                'total' => [
                    'uniqueVisitors' => 250000,
                ],
                'thisWeek' => [
                    'uniqueVisitors' => 786,
                    'growthPercent' => 25,
                ],
                'thisMonth' => [
                    'uniqueVisitors' => 2048,
                    'growthPercent' => 25,
                ]
            ]
        ];
    }
}
```

Note: The data returned from a data source that is used for a `ColumnView` need not adhere to a specific shape. The `viewOptions` configuration is used to extract data from arbitrary data shapes as you'll see below.

Example `viewOptions`:

```
'Vendor.Site:MyCustomNodeType':
  # ...
  ui:
    inspector:
      views:
        columnViewExample:
          label: 'ColumnView Example'
          group: examples
          view: 'Neos.Neos/Inspector/Views/Data/ColumnView'
          viewOptions:
            dataSource: vendor-site-column-view
            hero:
              data: total.uniqueVisitors
              label: 'visitors'
            columns:
              -
                data: thisWeek.uniqueVisitors
                label: 'this week'
              -
                data: thisMonth.uniqueVisitors
                label: 'this month'
              -
                data: thisMonth.growthPercent
                label: '% growth'
  # ...
```

`viewOptions` Reference:

`dataSource` (required string)

Identifier of a DataSource (given in the implementation class)

arguments (optional array)
If provided, the `arguments` will be passed as the second parameter to the data source's `getData` method.

hero (optional)
The hero element will be displayed with a larger font above all other data points.

data (required string or array)
A path to access the data from the data source. If given a string, the path will be split by `.`. The data this path points to should be numeric.

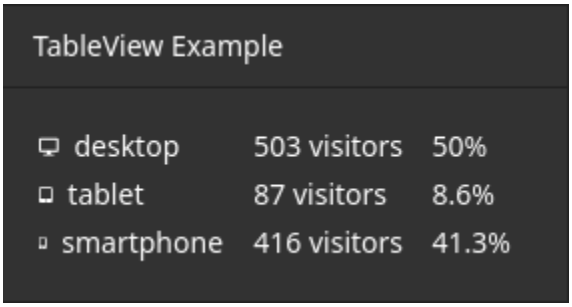
label (optional string)
A brief description of the number provided through `data`

columns (optional array)
This is a list of data points that will be displayed below the hero element. Each element in this array has the following options:

data (required string or array)
A path to access the data from the data source. If given a string, the path will be split by `.`. The data this path points to should be numeric.

label (optional string)
A brief description of the number provided through `data`

1.3.2 Data/TableView



The `TableView` displays data from a data source as a table.

A data source is a PHP class implementing `Neos\Neos\Service\DataSource\DataSourceInterface` (which is best achieved by extending `Neos\Neos\Service\DataSource\AbstractDataSource`). The example displayed above uses the following data source implementation:

```
<?php

/*
 * This script belongs to the package "Vendor.Site".
 */

declare(strict_types=1);

namespace Vendor\Site\Application\Neos\DataSource;

use Neos\Neos\Service\DataSource\AbstractDataSource;
use Neos\ContentRepository\Domain\Model\NodeInterface;
```

(continues on next page)

(continued from previous page)

```
final class TableViewDataSource extends AbstractDataSource
{
    /**
    * @var string
    */
    static protected $identifier = 'vendor-site-table-view';

    /**
    * Get data
    *
    * {@inheritdoc}
    */
    public function getData(NodeInterface $node = NULL, array $arguments = [])
    {
        return [
            'data' => [
                'rows' => [
                    [
                        'deviceCategory' => 'desktop',
                        'uniqueVisitors' => 503,
                        'percentage' => 50
                    ],
                    [
                        'deviceCategory' => 'tablet',
                        'uniqueVisitors' => 87,
                        'percentage' => 8.6
                    ],
                    [
                        'deviceCategory' => 'smartphone',
                        'uniqueVisitors' => 416,
                        'percentage' => 41.3
                    ],
                ],
            ]
        ];
    }
}
```

Note: The data returned from a data source that is used for a `TableView` need not adhere to a specific shape (as long as it represents some sort of list). The `viewOptions` configuration is used to extract data from arbitrary data shapes as you'll see below.

Example `viewOptions`:

```
'Vendor.Site:MyCustomNodeType':
  # ...
  ui:
    inspector:
      views:
        tableViewExample:
```

(continues on next page)

(continued from previous page)

```
        label: 'TableView Example'
        group: examples
        view: 'Neos.Neos/Inspector/Views/Data/TableView'
        viewOptions:
          dataSource: vendor-site-table-view
          collection: rows
          columns:
            -
              data: deviceCategory
              iconMap:
                desktop: icon-desktop
                tablet: icon-tablet
                smartphone: icon-mobile-phone
            -
              data: uniqueVisitors
              suffix: ' visitors'
            -
              data: percentage
              suffix: '%'
# ...
```

`viewOptions` Reference:

`dataSource` (required string)
Identifier of a DataSource (given in the implementation class)

`arguments` (optional array)
If provided, the `arguments` will be passed as the second parameter to the data source's `getData` method.

`collection` (required string or array)
A path to access the collection of interest from the data provided by the data source. If given a string, the path will be split by `.`. The data this path points to should be a numerically indexed array.

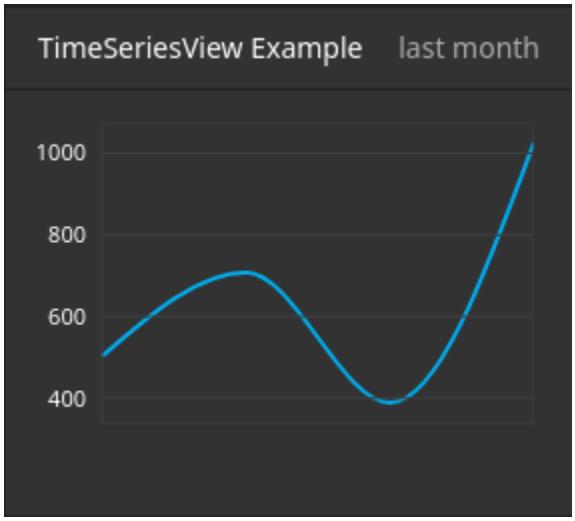
`columns` (optional array)
This key is used to configure the columns of the table. Each element in this array has the following options:

`data` (required string or array)
A path to access the data of an item of the list described by `collection`. If given a string, the path will be split by `.`.

`iconMap` (optional hashmap)
Maps the value provided through `data` to an icon that will be displayed in addition to the value.

`suffix` (optional string)
A brief description of the value provided through `data`

1.3.3 Data/TimeSeriesView



The `TimeSeriesView` displays data accumulated over time from a data source as a line chart.

A data source is a PHP class implementing `Neos\Neos\Service\DataSource\DataSourceInterface` (which is best achieved by extending `Neos\Neos\Service\DataSource\AbstractDataSource`). The example displayed above uses the following data source implementation:

```php
<?php

/*
 * This script belongs to the package "Vendor.Site".
 */

declare(strict_types=1);

namespace Vendor\Site\Application\Neos\DataSource;

use Neos\Neos\Service\DataSource\AbstractDataSource;
use Neos\ContentRepository\Domain\Model\NodeInterface;

final class TimeSeriesViewDataSource extends AbstractDataSource
{
    /**
    * @var string
    */
    static protected $identifier = 'vendor-site-time-series-view';

    /**
    * Get data
    *
    * {@inheritdoc}
    */
    public function getData(NodeInterface $node = NULL, array $arguments = [])
    {
        return [
            'data' => [
```

(continues on next page)

(continued from previous page)

```
                'rows' => [
                    [
                        'date' => new \DateTimeImmutable('2022-09-01'),
                        'uniqueVisitors' => 503
                    ],
                    [
                        'date' => new \DateTimeImmutable('2022-10-01'),
                        'uniqueVisitors' => 708
                    ],
                    [
                        'date' => new \DateTimeImmutable('2022-11-01'),
                        'uniqueVisitors' => 389
                    ],
                    [
                        'date' => new \DateTimeImmutable('2022-12-01'),
                        'uniqueVisitors' => 1027
                    ],
                ],
            ]
        ];
    }
}
```

Note: The data returned from a data source that is used for a `TimeSeriesView` need not adhere to a specific shape (as long as it includes a list of data associated with a time and date). The `viewOptions` configuration is used to extract data from arbitrary data shapes as you'll see below.

Example `viewOptions`:

```yaml
'Vendor.Site:MyCustomNodeType':
  # ...
  ui:
    inspector:
      views:
        timeSeriesViewExample:
          label: 'TimeSeriesView Example'
          group: examples
          view: 'Neos.Neos/Inspector/Views/Data/TimeSeriesView'
          viewOptions:
            subtitle: 'last month'
            dataSource: vendor-site-time-series-view
            collection: rows
            series:
              timeData: date
              valueData: uniqueVisitors
            chart:
              selectedInterval: weeks
              yAxisFromZero: false
  # ...
```

`viewOptions` Reference:

dataSource (required string)
Identifier of a DataSource (given in the implementation class)

arguments (optional array)
If provided, the `arguments` will be passed as the second parameter to the data source's `getData` method.

subtitle (optional string)
A brief description for the time series chart that will be displayed in the top right corner

collection (optional array or string)
A path to access the collection of interest from the data provided by the data source. If given a string, the path will be split by `.`. The data this path points to should be a numerically indexed array.

series (required object)
This object configures each data point that will be used to draw the chart.

timeData (required array or string)
A path to access the data of an item of the list described by `collection`. If given a string, the path will be split by `.`. The data this path points to should represent a time and date (e.g. an `integer`-typed unix timestamp or a `\DateTimeInterface`).

valueData (required array or string)
A path to access the data of an item of the list described by `collection`. If given a string, the path will be split by `.`. The data this path points to should be numeric.

chart (optional object)
This object configures the chart itself.

selectedInterval (optional, default="years")
This field configures the scale of the x-axis. Possible values are:

- `years` (or `Y`)
- `quarters` (or `Q`)
- `months` (or `M`)
- `weeks` (or `W`)
- `days` (or `D`)
- `seconds` (or `S`)

yAxisFromZero (optional boolean, default=false)
If set to `true` the y-axis will start from `0`. Otherwise the y-axis will start from the lowest available value.

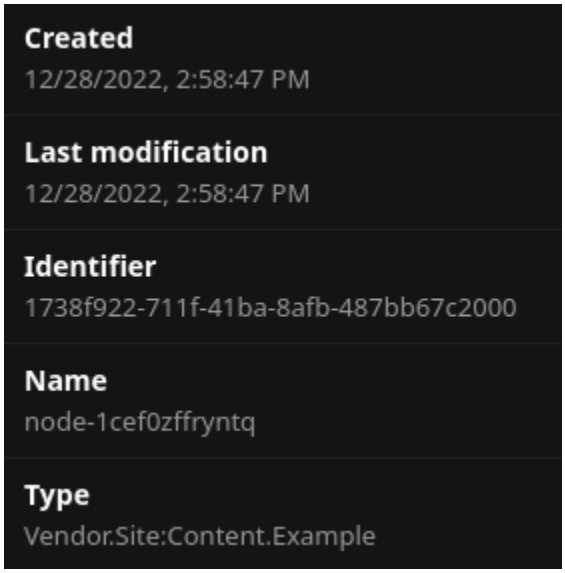
1.3.4 NodeInfoView

The `NodeInfoView` displays key characteristics of a node. In Neos, every node with a node type that extends `Neos.Neos:Node` (which applies to both content and document nodes by default), comes with a pre-configured `NodeInfoView`. It can be found in the Inspector under the `meta` tab.

It is unlikely, you'll ever need to configure this view manually. But if you need to, you can use a configuration like this one:

```
'Vendor.Site:MyCustomNodeType':
  # ...
  ui:
  inspector:
    views:
```

(continues on next page)



(continued from previous page)

```
    nodeInfoViewExample:
      label: 'NodeInfoView example'
      group: examples
      view: 'Neos.Neos/Inspector/Views/NodeInfoView'
# ...
```

The `NodeInfoView` has no `viewOptions`.

1.4 View Helper Reference

1.4.1 Content Repository ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

PaginateViewHelper

This ViewHelper renders a Pagination of nodes.

Implementation
Neos\ContentRepository\ViewHelpers\Widget\PaginateViewHelper

Arguments

- `widgetId` (string, *optional*): Unique identifier of the widget instance
- `as` (string): Variable name for the result set
- `parentNode` (NeosContentRepositoryDomainModelNodeInterface, *optional*): The parent node of the child nodes to show (instead of specifying the specific node set)
- `nodes` (array, *optional*): The specific collection of nodes to use for this paginator (instead of specifying the parentNode)
- `nodeTypeFilter` (string, *optional*): A node type (or more complex filter) to filter for in the results
- `configuration` (array, *optional*): Widget configuration

1.4.2 FluidAdaptor ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

f:debug

View helper that outputs its child nodes with NeosFlowvar_dump()

Implementation
Neos\FluidAdaptor\ViewHelpers\DebugViewHelper

Arguments

- `title` (string, *optional*): The title
- `typeOnly` (boolean, *optional*): Whether only the type should be returned instead of the whole chain.

Examples

inline notation and custom title:

```
{object -> f:debug(title: 'Custom title')}
```

Expected result:

```
all properties of {object} nicely highlighted (with custom title)
```

only output the type:

```
{object -> f:debug(typeOnly: true)}
```

Expected result:

```
the type or class name of {object}
```

f:flashMessages

View helper which renders the flash messages (if there are any) as an unsorted list.

Implementation
Neos\FluidAdaptor\ViewHelpers\FlashMessagesViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `as` (string, *optional*): The name of the current flashMessage variable for rendering inside
- `severity` (string, *optional*): severity of the messages (One of the NeosErrorMessagesMessage::SEVERITY_* constants)

Examples

Simple:

```
<f:flashMessages />
```

Expected result:

```
<ul>
  <li class="flashmessages-ok">Some Default Message</li>
  <li class="flashmessages-warning">Some Warning Message</li>
</ul>
```

Output with css class:

```
<f:flashMessages class="specialClass" />
```

Expected result:

```
<ul class="specialClass">
  <li class="specialClass-ok">Default Message</li>
  <li class="specialClass-notice"><h3>Some notice message</h3>With message title</li>
</ul>
```

Output flash messages as a list, with arguments and filtered by a severity:

```
<f:flashMessages severity="Warning" as="flashMessages">
        <dl class="messages">
        <f:for each="{flashMessages}" as="flashMessage">
                <dt>{flashMessage.code}</dt>
                <dd>{flashMessage}</dd>
        </f:for>
        </dl>
</f:flashMessages>
```

Expected result:

```
<dl class="messages">
        <dt>1013</dt>
        <dd>Some Warning Message.</dd>
</dl>
```

f:form

Used to output an HTML <form> tag which is targeted at the specified action, in the current controller and package.

Implementation
Neos\FluidAdaptor\ViewHelpers\FormViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `enctype` (string, *optional*): MIME type with which the form is submitted
- `method` (string, *optional*): Transfer type (GET or POST or dialog)
- `name` (string, *optional*): Name of form
- `onreset` (string, *optional*): JavaScript: On reset of the form
- `onsubmit` (string, *optional*): JavaScript: On submit of the form
- `action` (string, *optional*): Target action
- `arguments` (array, *optional*): Arguments
- `controller` (string, *optional*): Target controller. If NULL current controllerName is used
- `package` (string, *optional*): Target package. if NULL current package is used
- `subpackage` (string, *optional*): Target subpackage. if NULL current subpackage is used

- `object` (mixed, *optional*): object to use for the form. Use in conjunction with the "property" attribute on the sub tags
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (array, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `absolute` (boolean, *optional*): If set, an absolute action URI is rendered (only active if $actionUri is not set)
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `fieldNamePrefix` (string, *optional*): Prefix that will be added to all field names within this form
- `actionUri` (string, *optional*): can be used to overwrite the "action" attribute of the form tag
- `objectName` (string, *optional*): name of the object that is bound to this form. If this argument is not specified, the name attribute of this form is used to determine the FormObjectName
- `useParentRequest` (boolean, *optional*): If set, the parent Request will be used instead ob the current one
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Basic usage, POST method:

```
<f:form action="...">...</f:form>
```

Expected result:

```
<form action="...">...</form>
```

Basic usage, GET method:

```
<f:form action="..." method="get">...</f:form>
```

Expected result:

```
<form method="GET" action="...">...</form>
```

Form with a sepcified encoding type:

```
<f:form action="..." controller="..." package="..." enctype="multipart/form-data">...</
↪f:form>
```

Expected result:

```
<form enctype="multipart/form-data" action="...">...</form>
```

Binding a domain object to a form:

```
<f:form action="..." name="customer" object="{customer}">
  <f:form.hidden property="id" />
  <f:form.textfield property="name" />
</f:form>
```

Expected result:

```
A form where the value of {customer.name} is automatically inserted inside the textbox;␣
↪the name of the textbox is
set to match the property name.
```

f:form.button

Creates a button.

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\ButtonViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `autofocus` (string, *optional*): Specifies that a button should automatically get focus when the page loads
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `form` (string, *optional*): Specifies one or more forms the button belongs to
- `formaction` (string, *optional*): Specifies where to send the form-data when a form is submitted. Only for type="submit"
- `formenctype` (string, *optional*): Specifies how form-data should be encoded before sending it to a server. Only for type="submit" (e.g. "application/x-www-form-urlencoded", "multipart/form-data" or "text/plain")
- `formmethod` (string, *optional*): Specifies how to send the form-data (which HTTP method to use). Only for type="submit" (e.g. "get" or "post")

- `formnovalidate` (string, *optional*): Specifies that the form-data should not be validated on submission. Only for type="submit"
- `formtarget` (string, *optional*): Specifies where to display the response after submitting the form. Only for type="submit" (e.g. "_blank", "_self", "_parent", "_top", "framename")
- `type` (string, *optional*): Specifies the type of button (e.g. "button", "reset" or "submit")
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Defaults:

```
<f:form.button>Send Mail</f:form.button>
```

Expected result:

```
<button type="submit" name="" value="">Send Mail</button>
```

Disabled cancel button with some HTML5 attributes:

```
<f:form.button type="reset" name="buttonName" value="buttonValue" disabled="disabled"␣
↪formmethod="post" formnovalidate="formnovalidate">Cancel</f:form.button>
```

Expected result:

```
<button disabled="disabled" formmethod="post" formnovalidate="formnovalidate" type="reset
↪" name="myForm[buttonName]" value="buttonValue">Cancel</button>
```

f:form.checkbox

View Helper which creates a simple checkbox (<input type="checkbox">).

Implementation

Neos\FluidAdaptor\ViewHelpers\Form\CheckboxViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed): Value of input tag. Required for checkboxes
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `checked` (boolean, *optional*): Specifies that the input element should be preselected
- `multiple` (boolean, *optional*): Specifies whether this checkbox belongs to a multivalue (is part of a checkbox group)
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.checkbox name="myCheckBox" value="someValue" />
```

Expected result:

```
<input type="checkbox" name="myCheckBox" value="someValue" />
```

Preselect:

```
<f:form.checkbox name="myCheckBox" value="someValue" checked="{object.value} == 5" />
```

Expected result:

```
<input type="checkbox" name="myCheckBox" value="someValue" checked="checked" />
(depending on $object)
```

Bind to object property:

```
<f:form.checkbox property="interests" value="TYPO3" />
```

Expected result:

```
<input type="checkbox" name="user[interests][]" value="TYPO3" checked="checked" />
(depending on property "interests")
```

f:form.hidden

Renders an <input type="hidden" ...> tag.

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\HiddenViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.hidden name="myHiddenValue" value="42" />
```

Expected result:

```
<input type="hidden" name="myHiddenValue" value="42" />
```

f:form.password

View Helper which creates a simple Password Text Box (<input type="password">).

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\PasswordViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `required` (boolean, *optional*): Specifies that the input element requires a entry pre submit
- `maxlength` (int, *optional*): The maxlength attribute of the input field (will not be validated)
- `readonly` (string, *optional*): The readonly attribute of the input field
- `size` (int, *optional*): The size of the input field
- `placeholder` (string, *optional*): The placeholder of the input field
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.password name="myPassword" />
```

Expected result:

```
<input type="password" name="myPassword" value="default value" />
```

f:form.radio

View Helper which creates a simple radio button (<input type="radio">).

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\RadioViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed): Value of input tag. Required for radio buttons
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `checked` (boolean, *optional*): Specifies that the input element should be preselected
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.radio name="myRadioButton" value="someValue" />
```

Expected result:

```
<input type="radio" name="myRadioButton" value="someValue" />
```

Preselect:

```
<f:form.radio name="myRadioButton" value="someValue" checked="{object.value} == 5" />
```

Expected result:

```
<input type="radio" name="myRadioButton" value="someValue" checked="checked" />
(depending on $object)
```

Bind to object property:

```
<f:form.radio property="newsletter" value="1" /> yes
<f:form.radio property="newsletter" value="0" /> no
```

Expected result:

```
<input type="radio" name="user[newsletter]" value="1" checked="checked" /> yes
<input type="radio" name="user[newsletter]" value="0" /> no
(depending on property "newsletter")
```

f:form.select

This ViewHelper generates a <select> dropdown list for the use with a form.

Basic usage

The most straightforward way is to supply an associative array as the "options" parameter. The array key is used as option key, and the array value is used as human-readable name.

To pre-select a value, set "value" to the option key which should be selected. If the select box is a multi-select box (multiple="true"), then "value" can be an array as well.

Usage on domain objects

If you want to output domain objects, you can just pass them as array into the "options" parameter. To define what domain object value should be used as option key, use the "optionValueField" variable. Same goes for optionLabelField. If neither is given, the Identifier (UUID/uid) and the __toString() method are tried as fallbacks.

If the optionValueField variable is set, the getter named after that value is used to retrieve the option key. If the optionLabelField variable is set, the getter named after that value is used to retrieve the option value.

If the prependOptionLabel variable is set, an option item is added in first position, bearing an empty string or - if specified - the value of the prependOptionValue variable as value.

In the example below, the userArray is an array of "User" domain objects, with no array key specified. Thus the method $user->getId() is called to retrieve the key, and $user->getFirstName() to retrieve the displayed value of each entry. The "value" property now expects a domain object, and tests for object equivalence.

Translation of select content

The ViewHelper can be given a "translate" argument with configuration on how to translate option labels. The array can have the following keys: - "by" defines if translation by message id or original label is to be used ("id" or "label") - "using" defines if the option tag's "value" or "label" should be used as translation input, defaults to "value" - "locale" defines the locale identifier to use, optional, defaults to current locale - "source" defines the translation source name, optional, defaults to "Main" - "package" defines the package key of the translation source, optional, defaults to current package - "prefix" defines a prefix to use for the message id – only works in combination with "by id"

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\SelectViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="…">, "name" and "value" properties will be ignored.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `multiple` (string, *optional*): if set, multiple select field
- `size` (string, *optional*): Size of input field
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `required` (boolean, *optional*): Specifies that the select element requires at least one selected option
- `options` (array): Associative array with internal IDs as key, and the values are displayed in the select box
- `optionValueField` (string, *optional*): If specified, will call the appropriate getter on each object to determine the value.
- `optionLabelField` (string, *optional*): If specified, will call the appropriate getter on each object to determine the label.
- `sortByOptionLabel` (boolean, *optional*): If true, List will be sorted by label.
- `selectAllByDefault` (boolean, *optional*): If specified options are selected if none was set before.

- `errorClass` (string, *optional*): CSS class to set if there are errors for this ViewHelper
- `translate` (array, *optional*): Configures translation of ViewHelper output.
- `prependOptionLabel` (string, *optional*): If specified, will provide an option at first position with the specified label.
- `prependOptionValue` (string, *optional*): If specified, will provide an option at first position with the specified value. This argument is only respected if prependOptionLabel is set.

Examples

Basic usage:

```
<f:form.select name="paymentOptions" options="{payPal: 'PayPal International Services',␣
↪visa: 'VISA Card'}" />
```

Expected result:

```
<select name="paymentOptions">
  <option value="payPal">PayPal International Services</option>
  <option value="visa">VISA Card</option>
</select>
```

Preselect a default value:

```
<f:form.select name="paymentOptions" options="{payPal: 'PayPal International Services',␣
↪visa: 'VISA Card'}" value="visa" />
```

Expected result:

```
(Generates a dropdown box like above, except that "VISA Card" is selected.)
```

Use with domain objects:

```
<f:form.select name="users" options="{userArray}" optionValueField="id" optionLabelField=
↪"firstName" />
```

Expected result:

```
(Generates a dropdown box, using ids and first names of the User instances.)
```

Prepend a fixed option:

```
<f:form.select property="salutation" options="{salutations}" prependOptionLabel="-␣
↪select one -" />
```

Expected result:

```
<select name="salutation">
  <option value="">- select one -</option>
  <option value="Mr">Mr</option>
  <option value="Mrs">Mrs</option>
  <option value="Ms">Ms</option>
</select>
(depending on variable "salutations")
```

Label translation:

```
<f:form.select name="paymentOption" options="{payPal: 'PayPal International Services',␣
↪visa: 'VISA Card'}" translate="{by: 'id'}" />
```

Expected result:

```
(Generates a dropdown box and uses the values "payPal" and "visa" to look up
translations for those ids in the current package's "Main" XLIFF file.)
```

Label translation usign a prefix:

```
<f:form.select name="paymentOption" options="{payPal: 'PayPal International Services',␣
↪visa: 'VISA Card'}" translate="{by: 'id', prefix: 'shop.paymentOptions.'}" />
```

Expected result:

```
(Generates a dropdown box and uses the values "shop.paymentOptions.payPal"
and "shop.paymentOptions.visa" to look up translations for those ids in the
current package's "Main" XLIFF file.)
```

f:form.submit

Creates a submit button.

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\SubmitViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="…">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element

- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Defaults:

```
<f:form.submit value="Send Mail" />
```

Expected result:

```
<input type="submit" />
```

Dummy content for template preview:

```
<f:form.submit name="mySubmit" value="Send Mail"><button>dummy button</button></f:form.
↪submit>
```

Expected result:

```
<input type="submit" name="mySubmit" value="Send Mail" />
```

f:form.textarea

Textarea view helper. The value of the text area needs to be set via the "value" attribute, as with all other form ViewHelpers.

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\TextareaViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `rows` (int, *optional*): The number of rows of a text area
- `cols` (int, *optional*): The number of columns of a text area
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `required` (boolean, *optional*): If the field should be marked as required or not
- `placeholder` (string, *optional*): The placeholder of the textarea
- `autofocus` (string, *optional*): Specifies that a text area should automatically get focus when the page loads
- `maxlength` (int, *optional*): The maxlength attribute of the textarea (will not be validated)
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper

- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.textarea name="myTextArea" value="This is shown inside the textarea" />
```

Expected result:

```
<textarea name="myTextArea">This is shown inside the textarea</textarea>
```

f:form.textfield

View Helper which creates a text field (<input type="text">).

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\TextfieldViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `required` (boolean, *optional*): If the field should be marked as required or not
- `maxlength` (int, *optional*): The maxlength attribute of the input field (will not be validated)
- `readonly` (string, *optional*): The readonly attribute of the input field
- `size` (int, *optional*): The size of the input field
- `placeholder` (string, *optional*): The placeholder of the input field

- `autofocus` (string, *optional*): Specifies that a input field should automatically get focus when the page loads
- `type` (string, *optional*): The field type, e.g. "text", "email", "url" etc.
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.textfield name="myTextBox" value="default value" />
```

Expected result:

```
<input type="text" name="myTextBox" value="default value" />
```

f:form.upload

A view helper which generates an <input type="file"> HTML element. Make sure to set enctype="multipart/form-data" on the form!

If a file has been uploaded successfully and the form is re-displayed due to validation errors, this ViewHelper will render hidden fields that contain the previously generated resource so you won't have to upload the file again.

You can use a separate ViewHelper to display previously uploaded resources in order to remove/replace them.

Implementation
Neos\FluidAdaptor\ViewHelpers\Form\UploadViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag

- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (boolean, *optional*): Specifies that the input element should be disabled when the page loads
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `collection` (string, *optional*): Name of the resource collection this file should be uploaded to
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

Examples

Example:

```
<f:form.upload name="file" />
```

Expected result:

```
<input type="file" name="file" />
```

Multiple Uploads:

```
<f:form.upload property="attachments.0.originalResource" />
<f:form.upload property="attachments.1.originalResource" />
```

Expected result:

```
<input type="file" name="formObject[attachments][0][originalResource]">
<input type="file" name="formObject[attachments][0][originalResource]">
```

Default resource:

```
<f:form.upload name="file" value="{someDefaultResource}" />
```

Expected result:

```
<input type="hidden" name="file[originallySubmittedResource][__identity]" value="
↪<someDefaultResource-UUID>" />
<input type="file" name="file" />
```

Specifying the resource collection for the new resource:

```
<f:form.upload name="file" collection="invoices"/>
```

Expected result:

```
<input type="file" name="yourInvoice" />
<input type="hidden" name="yourInvoice[__collectionName]" value="invoices" />
```

f:format.base64Decode

Applies base64_decode to the input

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\Base64DecodeViewHelper

Arguments

- `value` (string, *optional*): string to format

f:format.bytes

Formats an integer with a byte count into human-readable form.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\BytesViewHelper

Arguments

- `forceLocale` (mixed, *optional*): Whether if, and what, Locale should be used. May be boolean, string or NeosFlowI18nLocale
- `value` (integer, *optional*): The incoming data to convert, or NULL if VH children should be used
- `decimals` (integer, *optional*): The number of digits after the decimal point
- `decimalSeparator` (string, *optional*): The decimal point character
- `thousandsSeparator` (string, *optional*): The character for grouping the thousand digits
- `localeFormatLength` (string, *optional*): Format length if locale set in $forceLocale. Must be one of NeosFlowI18nCldrReaderNumbersReader::FORMAT_LENGTH_*'s constants.

Examples

Defaults:

```
{fileSize -> f:format.bytes()}
```

Expected result:

```
123 KB
// depending on the value of {fileSize}
```

With all parameters:

```
{fileSize -> f:format.bytes(decimals: 2, decimalSeparator: ',', thousandsSeparator: ',')}
```

Expected result:

```
1,023.00 B
// depending on the value of {fileSize}
```

Inline notation with current locale used:

```
{fileSize -> f:format.bytes(forceLocale: true)}
```

Expected result:

```
6.543,21 KB
// depending on the value of {fileSize} and the current locale
```

Inline notation with specific locale used:

```
{fileSize -> f:format.bytes(forceLocale: 'de_CH')}
```

Expected result:

```
1'337.42 MB
// depending on the value of {fileSize}
```

f:format.case

Modifies the case of an input string to upper- or lowercase or capitalization. The default transformation will be upper-case as in `mb_convert_case` [1].

Possible modes are:

`lower`
: Transforms the input string to its lowercase representation

`upper`
: Transforms the input string to its uppercase representation

`capital`
: Transforms the input string to its first letter upper-cased, i.e. capitalization

`uncapital`
: Transforms the input string to its first letter lower-cased, i.e. uncapitalization

`capitalWords`
: Transforms the input string to each containing word being capitalized

Note that the behavior will be the same as in the appropriate PHP function `mb_convert_case` [1]; especially regarding locale and multibyte behavior.

Implementation
: Neos\FluidAdaptor\ViewHelpers\Format\CaseViewHelper

Arguments

- `value` (string, *optional*): The input value. If not given, the evaluated child nodes will be used
- `mode` (string, *optional*): The case to apply, must be one of this' CASE_* constants. Defaults to uppercase application

f:format.crop

Use this view helper to crop the text between its opening and closing tags.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\CropViewHelper

Arguments

- `maxCharacters` (integer): Place where to truncate the string
- `append` (string, *optional*): What to append, if truncation happened
- `value` (string, *optional*): The input value which should be cropped. If not set, the evaluated contents of the child nodes will be used

Examples

Defaults:

```
<f:format.crop maxCharacters="10">This is some very long text</f:format.crop>
```

Expected result:

```
This is so...
```

Custom suffix:

```
<f:format.crop maxCharacters="17" append=" [more]">This is some very long text</f:format.
↪crop>
```

Expected result:

```
This is some very [more]
```

Inline notation:

```
<span title="Location: {user.city -> f:format.crop(maxCharacters: '12')}">John Doe</span>
```

Expected result:

```
<span title="Location: Newtownmount...">John Doe</span>
```

f:format.currency

Formats a given float to a currency representation.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\CurrencyViewHelper

Arguments

- `forceLocale` (mixed, *optional*): Whether if, and what, Locale should be used. May be boolean, string or NeosFlowI18nLocale
- `currencySign` (string, *optional*): (optional) The currency sign, eg $ or €.
- `decimalSeparator` (string, *optional*): (optional) The separator for the decimal point.
- `thousandsSeparator` (string, *optional*): (optional) The thousands separator.
- `prependCurrency` (boolean, *optional*): (optional) Indicates if currency symbol should be placed before or after the numeric value.
- `separateCurrency` (boolean, *optional*): (optional) Indicates if a space character should be placed between the number and the currency sign.
- `decimals` (integer, *optional*): (optional) The number of decimal places.
- `currencyCode` (string, *optional*): (optional) The ISO 4217 currency code of the currency to format. Used to set decimal places and rounding.

Examples

Defaults:

```
<f:format.currency>123.456</f:format.currency>
```

Expected result:

```
123,46
```

All parameters:

```
<f:format.currency currencySign="$" decimalSeparator="." thousandsSeparator=","␣
↪prependCurrency="false", separateCurrency="true", decimals="2">54321</f:format.
↪currency>
```

Expected result:

```
54,321.00 $
```

Inline notation:

```
{someNumber -> f:format.currency(thousandsSeparator: ',', currencySign: '€')}
```

Expected result:

```
54,321,00 €
(depending on the value of {someNumber})
```

Inline notation with current locale used:

```
{someNumber -> f:format.currency(currencySign: '€', forceLocale: true)}
```

Expected result:

```
54.321,00 €
(depending on the value of {someNumber} and the current locale)
```

Inline notation with specific locale used:

```
{someNumber -> f:format.currency(currencySign: 'EUR', forceLocale: 'de_DE')}
```

Expected result:

```
54.321,00 EUR
(depending on the value of {someNumber})
```

Inline notation with different position for the currency sign:

```
{someNumber -> f:format.currency(currencySign: '€', prependCurrency: 'true')}
```

Expected result:

```
€ 54.321,00
(depending on the value of {someNumber})
```

Inline notation with no space between the currency and no decimal places:

```
{someNumber -> f:format.currency(currencySign: '€', separateCurrency: 'false', decimals:
↪'0')}
```

Expected result:

```
54.321€
(depending on the value of {someNumber})
```

f:format.date

Formats a DateTime object.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\DateViewHelper

Arguments

- `forceLocale` (mixed, *optional*): Whether if, and what, Locale should be used. May be boolean, string or NeosFlowI18nLocale
- `date` (mixed, *optional*): either a DateTime object or a string that is accepted by DateTime constructor
- `format` (string, *optional*): Format String which is taken to format the Date/Time if none of the locale options are set.
- `localeFormatType` (string, *optional*): Whether to format (according to locale set in $forceLocale) date, time or datetime. Must be one of NeosFlowI18nCldrReaderDatesReader::FORMAT_TYPE_*'s constants.

- `localeFormatLength` (string, *optional*): Format length if locale set in $forceLocale. Must be one of NeosFlowI18nCldrReaderDatesReader::FORMAT_LENGTH_*'s constants.
- `cldrFormat` (string, *optional*): Format string in CLDR format (see http://cldr.unicode.org/translation/date-time)

Examples

Defaults:

```
<f:format.date>{dateObject}</f:format.date>
```

Expected result:

```
1980-12-13
(depending on the current date)
```

Custom date format:

```
<f:format.date format="H:i">{dateObject}</f:format.date>
```

Expected result:

```
01:23
(depending on the current time)
```

strtotime string:

```
<f:format.date format="d.m.Y - H:i:s">+1 week 2 days 4 hours 2 seconds</f:format.date>
```

Expected result:

```
13.12.1980 - 21:03:42
(depending on the current time, see http://www.php.net/manual/en/function.strtotime.php)
```

output date from unix timestamp:

```
<f:format.date format="d.m.Y - H:i:s">@{someTimestamp}</f:format.date>
```

Expected result:

```
13.12.1980 - 21:03:42
(depending on the current time. Don't forget the "@" in front of the timestamp see http:/
↪/www.php.net/manual/en/function.strtotime.php)
```

Inline notation:

```
{f:format.date(date: dateObject)}
```

Expected result:

```
1980-12-13
(depending on the value of {dateObject})
```

Inline notation (2nd variant):

```
{dateObject -> f:format.date()}
```

Expected result:

```
1980-12-13
(depending on the value of {dateObject})
```

Inline notation, outputting date only, using current locale:

```
{dateObject -> f:format.date(localeFormatType: 'date', forceLocale: true)}
```

Expected result:

```
13.12.1980
(depending on the value of {dateObject} and the current locale)
```

Inline notation with specific locale used:

```
{dateObject -> f:format.date(forceLocale: 'de_DE')}
```

Expected result:

```
13.12.1980 11:15:42
(depending on the value of {dateObject})
```

f:format.htmlentities

Applies htmlentities() escaping to a value

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\HtmlentitiesViewHelper

Arguments

- `value` (string, *optional*): string to format
- `keepQuotes` (boolean, *optional*): if true, single and double quotes won't be replaced (sets ENT_NOQUOTES flag)
- `encoding` (string, *optional*): the encoding format
- `doubleEncode` (string, *optional*): If false existing html entities won't be encoded, the default is to convert everything.

f:format.htmlentitiesDecode

Applies html_entity_decode() to a value

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\HtmlentitiesDecodeViewHelper

Arguments

- `value` (string, *optional*): string to format
- `keepQuotes` (boolean, *optional*): if true, single and double quotes won't be replaced (sets ENT_NOQUOTES flag)
- `encoding` (string, *optional*): the encoding format

f:format.identifier

This ViewHelper renders the identifier of a persisted object (if it has an identity). Usually the identifier is the UUID of the object, but it could be an array of the identity properties, too.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\IdentifierViewHelper

Arguments

- `value` (object, *optional*): the object to render the identifier for, or NULL if VH children should be used

f:format.json

Wrapper for PHPs json_encode function.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\JsonViewHelper

Arguments

- `value` (mixed, *optional*): The incoming data to convert, or NULL if VH children should be used
- `forceObject` (boolean, *optional*): Outputs an JSON object rather than an array

Examples

encoding a view variable:

```
{someArray -> f:format.json()}
```

Expected result:

```
["array","values"]
// depending on the value of {someArray}
```

associative array:

```
{f:format.json(value: {foo: 'bar', bar: 'baz'})}
```

Expected result:

```
{"foo":"bar","bar":"baz"}
```

non-associative array with forced object:

```
{f:format.json(value: {0: 'bar', 1: 'baz'}, forceObject: true)}
```

Expected result:

```
{"0":"bar","1":"baz"}
```

f:format.nl2br

Wrapper for PHPs nl2br function.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\Nl2brViewHelper

Arguments

- `value` (string, *optional*): string to format

f:format.number

Formats a number with custom precision, decimal point and grouped thousands.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\NumberViewHelper

Arguments

- `forceLocale` (mixed, *optional*): Whether if, and what, Locale should be used. May be boolean, string or NeosFlowI18nLocale
- `decimals` (integer, *optional*): The number of digits after the decimal point
- `decimalSeparator` (string, *optional*): The decimal point character
- `thousandsSeparator` (string, *optional*): The character for grouping the thousand digits
- `localeFormatLength` (string, *optional*): Format length if locale set in $forceLocale. Must be one of NeosFlowI18nCldrReaderNumbersReader::FORMAT_LENGTH_*'s constants.

f:format.padding

Formats a string using PHPs str_pad function.

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\PaddingViewHelper

Arguments

- `padLength` (integer): Length of the resulting string. If the value of pad_length is negative or less than the length of the input string, no padding takes place.
- `padString` (string, *optional*): The padding string
- `padType` (string, *optional*): Append the padding at this site (Possible values: right,left,both. Default: right)
- `value` (string, *optional*): string to format

f:format.stripTags

Removes tags from the given string (applying PHPs strip_tags() function)

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\StripTagsViewHelper

Arguments

- `value` (string, *optional*): string to format

f:format.urlencode

Encodes the given string according to http://www.faqs.org/rfcs/rfc3986.html (applying PHPs rawurlencode() function)

Implementation
Neos\FluidAdaptor\ViewHelpers\Format\UrlencodeViewHelper

Arguments

- `value` (string, *optional*): string to format

f:link.action

A view helper for creating links to actions.

Implementation
Neos\FluidAdaptor\ViewHelpers\Link\ActionViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor
- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document
- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `action` (string): Target action
- `arguments` (array, *optional*): Arguments
- `controller` (string, *optional*): Target controller. If NULL current controllerName is used
- `package` (string, *optional*): Target package. if NULL current package is used
- `subpackage` (string, *optional*): Target subpackage. if NULL current subpackage is used
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (array, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `useParentRequest` (boolean, *optional*): If set, the parent Request will be used instead of the current one. Note: using this argument can be a sign of undesired tight coupling, use with care
- `absolute` (boolean, *optional*): By default this ViewHelper renders links with absolute URIs. If this is false, a relative URI is created instead
- `useMainRequest` (boolean, *optional*): If set, the main Request will be used instead of the current one. Note: using this argument can be a sign of undesired tight coupling, use with care

Examples

Defaults:

```
<f:link.action>some link</f:link.action>
```

Expected result:

```
<a href="currentpackage/currentcontroller">some link</a>
(depending on routing setup and current package/controller/action)
```

Additional arguments:

```
<f:link.action action="myAction" controller="MyController" package="YourCompanyName.
↪MyPackage" subpackage="YourCompanyName.MySubpackage" arguments="{key1: 'value1', key2:
↪'value2'}">some link</f:link.action>
```

Expected result:

```
<a href="mypackage/mycontroller/mysubpackage/myaction?key1=value1&amp;key2=value2">some␣
↪link</a>
(depending on routing setup)
```

f:link.email

Email link view helper. Generates an email link.

Implementation
Neos\FluidAdaptor\ViewHelpers\Link\EmailViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor
- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document

- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `email` (string): The email address to be turned into a link.

Examples

basic email link:

```
<f:link.email email="foo@bar.tld" />
```

Expected result:

```
<a href="mailto:foo@bar.tld">foo@bar.tld</a>
```

Email link with custom linktext:

```
<f:link.email email="foo@bar.tld">some custom content</f:link.email>
```

Expected result:

```
<a href="mailto:foo@bar.tld">some custom content</a>
```

f:link.external

A view helper for creating links to external targets.

Implementation
Neos\FluidAdaptor\ViewHelpers\Link\ExternalViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor

- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document
- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `uri` (string): the URI that will be put in the href attribute of the rendered link tag
- `defaultScheme` (string, *optional*): scheme the href attribute will be prefixed with if specified $uri does not contain a scheme already

Examples

custom default scheme:

```
<f:link.external uri="neos.io" defaultScheme="sftp">external ftp link</f:link.external>
```

Expected result:

```
<a href="sftp://neos.io">external ftp link</a>
```

f:renderChildren

Render the inner parts of a Widget. This ViewHelper can only be used in a template which belongs to a Widget Controller.

It renders everything inside the Widget ViewHelper, and you can pass additional arguments.

Implementation
Neos\FluidAdaptor\ViewHelpers\RenderChildrenViewHelper

Arguments

- `arguments` (array, *optional*): Arguments to pass to the rendering

Examples

Basic usage:

```
<!-- in the widget template -->
Header
<f:renderChildren arguments="{foo: 'bar'}" />
Footer

<-- in the outer template, using the widget -->

<x:widget.someWidget>
  Foo: {foo}
</x:widget.someWidget>
```

Expected result:

```
Header
Foo: bar
Footer
```

f:security.csrfToken

ViewHelper that outputs a CSRF token which is required for "unsafe" requests (e.g. POST, PUT, DELETE, ...).

Note: You won't need this ViewHelper if you use the Form ViewHelper, because that creates a hidden field with the CSRF token for unsafe requests automatically. This ViewHelper is mainly useful in conjunction with AJAX.

Implementation
Neos\FluidAdaptor\ViewHelpers\Security\CsrfTokenViewHelper

f:security.ifAccess

This view helper implements an ifAccess/else condition.

Implementation
Neos\FluidAdaptor\ViewHelpers\Security\IfAccessViewHelper

Arguments

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `condition` (boolean, *optional*): Condition expression conforming to Fluid boolean rules
- `privilegeTarget` (string): Condition expression conforming to Fluid boolean rules
- `parameters` (array, *optional*): Condition expression conforming to Fluid boolean rules

f:security.ifAuthenticated

This view helper implements an ifAuthenticated/else condition.

Implementation
Neos\FluidAdaptor\ViewHelpers\Security\IfAuthenticatedViewHelper

Arguments

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `condition` (boolean, *optional*): Condition expression conforming to Fluid boolean rules

f:security.ifHasRole

This view helper implements an ifHasRole/else condition.

Implementation
Neos\FluidAdaptor\ViewHelpers\Security\IfHasRoleViewHelper

Arguments

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `condition` (boolean, *optional*): Condition expression conforming to Fluid boolean rules
- `role` (mixed): The role or role identifier.
- `packageKey` (string, *optional*): PackageKey of the package defining the role.
- `account` (NeosFlowSecurityAccount, *optional*): If specified, this subject of this check is the given Account instead of the currently authenticated account

f:translate

Returns translated message using source message or key ID.

Also replaces all placeholders with formatted versions of provided values.

Implementation
Neos\FluidAdaptor\ViewHelpers\TranslateViewHelper

Arguments

- `id` (string, *optional*): Id to use for finding translation (trans-unit id in XLIFF)
- `value` (string, *optional*): If $key is not specified or could not be resolved, this value is used. If this argument is not set, child nodes will be used to render the default
- `arguments` (array, *optional*): Numerically indexed array of values to be inserted into placeholders
- `source` (string, *optional*): Name of file with translations (use / as a directory separator)
- `package` (string, *optional*): Target package key. If not set, the current package key will be used
- `quantity` (mixed, *optional*): A number to find plural form for (float or int), NULL to not use plural forms
- `locale` (string, *optional*): An identifier of locale to use (NULL for use the default locale)

Examples

Translation by id:

```
<f:translate id="user.unregistered">Unregistered User</f:translate>
```

Expected result:

```
translation of label with the id "user.unregistered" and a fallback to "Unregistered User
↪"
```

Inline notation:

```
{f:translate(id: 'some.label.id', value: 'fallback result')}
```

Expected result:

```
translation of label with the id "some.label.id" and a fallback to "fallback result"
```

Custom source and locale:

```
<f:translate id="some.label.id" source="LabelsCatalog" locale="de_DE"/>
```

Expected result:

```
translation from custom source "SomeLabelsCatalog" for locale "de_DE"
```

Custom source from other package:

```
<f:translate id="some.label.id" source="LabelsCatalog" package="OtherPackage"/>
```

Expected result:

```
translation from custom source "LabelsCatalog" in "OtherPackage"
```

Arguments:

```
<f:translate arguments="{0: 'foo', 1: '99.9'}"><![CDATA[Untranslated {0} and {1,number}
↪]]></f:translate>
```

Expected result:

```
translation of the label "Untranslated foo and 99.9"
```

Translation by label:

```
<f:translate>Untranslated label</f:translate>
```

Expected result:

```
translation of the label "Untranslated label"
```

f:uri.action

A view helper for creating URIs to actions.

Implementation
Neos\FluidAdaptor\ViewHelpers\Uri\ActionViewHelper

Arguments

- `action` (string): Target action
- `arguments` (array, *optional*): Arguments
- `controller` (string, *optional*): Target controller. If NULL current controllerName is used
- `package` (string, *optional*): Target package. if NULL current package is used
- `subpackage` (string, *optional*): Target subpackage. if NULL current subpackage is used
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (array, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `absolute` (boolean, *optional*): By default this ViewHelper renders links with absolute URIs. If this is false, a relative URI is created instead
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `useParentRequest` (boolean, *optional*): If set, the parent Request will be used instead of the current one. Note: using this argument can be a sign of undesired tight coupling, use with care
- `useMainRequest` (boolean, *optional*): If set, the main Request will be used instead of the current one. Note: using this argument can be a sign of undesired tight coupling, use with care

Examples

Defaults:

```
<f:uri.action>some link</f:uri.action>
```

Expected result:

```
currentpackage/currentcontroller
(depending on routing setup and current package/controller/action)
```

Additional arguments:

```
<f:uri.action action="myAction" controller="MyController" package="YourCompanyName.
↪MyPackage" subpackage="YourCompanyName.MySubpackage" arguments="{key1: 'value1', key2:
↪'value2'}">some link</f:uri.action>
```

Expected result:

```
mypackage/mycontroller/mysubpackage/myaction?key1=value1&amp;key2=value2
(depending on routing setup)
```

f:uri.email

Email uri view helper. Currently the specified email is simply prepended by "mailto:" but we might add spam protection.

Implementation
Neos\FluidAdaptor\ViewHelpers\Uri\EmailViewHelper

Arguments

- `email` (string): The email address to be turned into a mailto uri.

Examples

basic email uri:

```
<f:uri.email email="foo@bar.tld" />
```

Expected result:

```
mailto:foo@bar.tld
```

f:uri.external

A view helper for creating URIs to external targets. Currently the specified URI is simply passed through.

Implementation
Neos\FluidAdaptor\ViewHelpers\Uri\ExternalViewHelper

Arguments

- `uri` (string): target URI
- `defaultScheme` (string, *optional*): target URI

Examples

custom default scheme:

```
<f:uri.external uri="neos.io" defaultScheme="sftp" />
```

Expected result:

```
sftp://neos.io
```

f:uri.resource

A view helper for creating URIs to resources.

Implementation
Neos\FluidAdaptor\ViewHelpers\Uri\ResourceViewHelper

Arguments

- `path` (string, *optional*): Location of the resource, can be either a path relative to the Public resource directory of the package or a resource://... URI
- `package` (string, *optional*): Target package key. If not set, the current package key will be used
- `resource` (NeosFlowResourceManagementPersistentResource, *optional*): If specified, this resource object is used instead of the path and package information
- `localize` (bool, *optional*): Whether resource localization should be attempted or not.

Examples

Defaults:

```
<link href="{f:uri.resource(path: 'CSS/Stylesheet.css')}" rel="stylesheet" />
```

Expected result:

```
<link href="http://yourdomain.tld/_Resources/Static/YourPackage/CSS/Stylesheet.css" rel=
↪"stylesheet" />
(depending on current package)
```

Other package resource:

```
{f:uri.resource(path: 'gfx/SomeImage.png', package: 'DifferentPackage')}
```

Expected result:

```
http://yourdomain.tld/_Resources/Static/DifferentPackage/gfx/SomeImage.png
(depending on domain)
```

Static resource URI:

```
{f:uri.resource(path: 'resource://DifferentPackage/Public/gfx/SomeImage.png')}
```

Expected result:

```
http://yourdomain.tld/_Resources/Static/DifferentPackage/gfx/SomeImage.png
(depending on domain)
```

Persistent resource object:

```
<img src="{f:uri.resource(resource: myImage.resource)}" />
```

Expected result:

```
<img src="http://yourdomain.tld/_Resources/Persistent/
↪69e73da3ce0ad08c717b7b9f1c759182d6650944.jpg" />
(depending on your resource object)
```

f:validation.ifHasErrors

This view helper allows to check whether validation errors adhere to the current request.

Implementation
Neos\FluidAdaptor\ViewHelpers\Validation\IfHasErrorsViewHelper

Arguments

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `for` (string, *optional*): The argument or property name or path to check for error(s). If not set any validation error leads to the "then child" to be rendered

f:validation.results

Validation results view helper

Implementation
Neos\FluidAdaptor\ViewHelpers\Validation\ResultsViewHelper

Arguments

- `for` (string, *optional*): The name of the error name (e.g. argument name or property name). This can also be a property path (like blog.title), and will then only display the validation errors of that property.
- `as` (string, *optional*): The name of the variable to store the current error

Examples

Output error messages as a list:

```
<f:validation.results>
  <f:if condition="{validationResults.flattenedErrors}">
    <ul class="errors">
      <f:for each="{validationResults.flattenedErrors}" as="errors" key="propertyPath">
        <li>{propertyPath}
          <ul>
          <f:for each="{errors}" as="error">
            <li>{error.code}: {error}</li>
          </f:for>
          </ul>
        </li>
      </f:for>
```

(continues on next page)

(continued from previous page)

```
    </ul>
  </f:if>
</f:validation.results>
```

Expected result:

```
<ul class="errors">
  <li>1234567890: Validation errors for argument "newBlog"</li>
</ul>
```

Output error messages for a single property:

```
<f:validation.results for="someProperty">
  <f:if condition="{validationResults.flattenedErrors}">
    <ul class="errors">
      <f:for each="{validationResults.errors}" as="error">
        <li>{error.code}: {error}</li>
      </f:for>
    </ul>
  </f:if>
</f:validation.results>
```

Expected result:

```
<ul class="errors">
  <li>1234567890: Some error message</li>
</ul>
```

f:widget.autocomplete

Usage: <f:input id="name" ... /> <f:widget.autocomplete for="name" objects="{posts}" searchProperty="author">

Make sure to include jQuery and jQuery UI in the HTML, like that:

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script> <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.4/jquery-ui.min.js"></script> <link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.3/themes/base/jquery-ui.css" type="text/css" media="all" /> <link rel="stylesheet" href="http://static.jquery.com/ui/css/demo-docs-theme/ui.theme.css" type="text/css" media="all" />

Implementation

Neos\FluidAdaptor\ViewHelpers\Widget\AutocompleteViewHelper

Arguments

- `widgetId` (string, *optional*): Unique identifier of the widget instance
- `objects` (NeosFlowPersistenceQueryResultInterface): Objects
- `for` (string): for
- `searchProperty` (string): Property to search
- `configuration` (array, *optional*): Widget configuration

f:widget.link

widget.link ViewHelper This ViewHelper can be used inside widget templates in order to render links pointing to widget actions

Implementation
Neos\FluidAdaptor\ViewHelpers\Widget\LinkViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor
- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document
- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `action` (string, *optional*): Target action
- `arguments` (array, *optional*): Arguments
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `ajax` (boolean, *optional*): true if the URI should be to an AJAX widget, false otherwise
- `includeWidgetContext` (boolean, *optional*): true if the URI should contain the serialized widget context (only useful for stateless AJAX widgets)

f:widget.paginate

This ViewHelper renders a Pagination of objects.

Implementation
Neos\FluidAdaptor\ViewHelpers\Widget\PaginateViewHelper

Arguments

- `widgetId` (string, *optional*): Unique identifier of the widget instance
- `objects` (NeosFlowPersistenceQueryResultInterface): Objects
- `as` (string): as
- `configuration` (array, *optional*): Widget configuration

f:widget.uri

widget.uri ViewHelper This ViewHelper can be used inside widget templates in order to render URIs pointing to widget actions

Implementation
Neos\FluidAdaptor\ViewHelpers\Widget\UriViewHelper

Arguments

- `action` (string): Target action
- `arguments` (array, *optional*): Arguments
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `ajax` (boolean, *optional*): true if the URI should be to an AJAX widget, false otherwise
- `includeWidgetContext` (boolean, *optional*): true if the URI should contain the serialized widget context (only useful for stateless AJAX widgets)

1.4.3 Form ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

neos.form:form

Custom form ViewHelper that renders the form state instead of referrer fields

Implementation
Neos\Form\ViewHelpers\FormViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `enctype` (string, *optional*): MIME type with which the form is submitted
- `method` (string, *optional*): Transfer type (GET or POST or dialog)
- `name` (string, *optional*): Name of form
- `onreset` (string, *optional*): JavaScript: On reset of the form
- `onsubmit` (string, *optional*): JavaScript: On submit of the form
- `action` (string, *optional*): Target action
- `arguments` (array, *optional*): Arguments
- `controller` (string, *optional*): Target controller. If NULL current controllerName is used
- `package` (string, *optional*): Target package. if NULL current package is used
- `subpackage` (string, *optional*): Target subpackage. if NULL current subpackage is used
- `object` (mixed, *optional*): object to use for the form. Use in conjunction with the "property" attribute on the sub tags
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (array, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `absolute` (boolean, *optional*): If set, an absolute action URI is rendered (only active if $actionUri is not set)
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `fieldNamePrefix` (string, *optional*): Prefix that will be added to all field names within this form
- `actionUri` (string, *optional*): can be used to overwrite the "action" attribute of the form tag
- `objectName` (string, *optional*): name of the object that is bound to this form. If this argument is not specified, the name attribute of this form is used to determine the FormObjectName
- `useParentRequest` (boolean, *optional*): If set, the parent Request will be used instead ob the current one
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element

- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

neos.form:form.datePicker

Display a jQuery date picker.

Note: Requires jQuery UI to be included on the page.

Implementation
Neos\Form\ViewHelpers\Form\DatePickerViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="…">, "name" and "value" properties will be ignored.
- `size` (int, *optional*): The size of the input field
- `placeholder` (string, *optional*): Specifies a short hint that describes the expected value of an input element
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `initialDate` (string, *optional*): Initial date (@see http://www.php.net/manual/en/datetime.formats.php for supported formats)
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `dateFormat` (string, *optional*): Format to use for date formatting
- `enableDatePicker` (boolean, *optional*): true to enable a date picker

neos.form:form.formElementRootlinePath

Form Element Rootline Path

Implementation
Neos\Form\ViewHelpers\Form\FormElementRootlinePathViewHelper

Arguments

- `renderable` (NeosFormCoreModelRenderableRenderableInterface): The renderable

neos.form:form.timePicker

Displays two select-boxes for hour and minute selection.

Implementation
Neos\Form\ViewHelpers\Form\TimePickerViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="…">, "name" and "value" properties will be ignored.
- `size` (int, *optional*): The size of the select field
- `placeholder` (string, *optional*): Specifies a short hint that describes the expected value of an input element
- `disabled` (string, *optional*): Specifies that the select element should be disabled when the page loads
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `initialDate` (string, *optional*): Initial time (@see http://www.php.net/manual/en/datetime.formats.php for supported formats)
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event

neos.form:form.uploadedImage

This ViewHelper makes the specified Image object available for its childNodes. In case the form is redisplayed because of validation errors, a previously uploaded image will be correctly used.

Implementation
Neos\Form\ViewHelpers\Form\UploadedImageViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `as` (string, *optional*): Variable name to use for the uploaded image

Examples

Example:

```
<f:form.upload property="image" />
<c:form.uploadedImage property="image" as="theImage">
  <a href="{f:uri.resource(resource: theImage.resource)}">Link to image resource</a>
</c:form.uploadedImage>
```

Expected result:

```
<a href="...">Link to image resource</a>
```

neos.form:form.uploadedResource

This ViewHelper makes the specified PersistentResource available for its childNodes. If no resource object was found at the specified position, the child nodes are not rendered.

In case the form is redisplayed because of validation errors, a previously uploaded resource will be correctly used.

Implementation
Neos\Form\ViewHelpers\Form\UploadedResourceViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed, *optional*): Value of input tag
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `as` (string, *optional*): Variable name to use for the uploaded resource

Examples

Example:

```
<f:form.upload property="file" />
<c:form.uploadedResource property="file" as="theResource">
  <a href="{f:uri.resource(resource: theResource)}">Link to resource</a>
</c:form.uploadedResource>
```

Expected result:

```
<a href="...">Link to resource</a>
```

neos.form:render

Main Entry Point to render a Form into a Fluid Template

<pre> {namespace form=NeosFormViewHelpers} <form:render factoryClass="NameOfYourCustomFactoryClass" /> </pre>

The factory class must implement {@link NeosFormFactoryFormFactoryInterface}.

Implementation
Neos\Form\ViewHelpers\RenderViewHelper

Arguments

- `persistenceIdentifier` (string, *optional*): The persistence identifier for the form
- `factoryClass` (string, *optional*): The fully qualified class name of the factory (which has to implement NeosFormFactoryFormFactoryInterface)
- `presetName` (string, *optional*): Name of the preset to use
- `overrideConfiguration` (array, *optional*): Factory specific configuration

neos.form:renderHead

Output the configured stylesheets and JavaScript include tags for a given preset

Implementation
Neos\Form\ViewHelpers\RenderHeadViewHelper

Arguments

- `presetName` (string, *optional*): Relative Fusion path to be rendered

neos.form:renderRenderable

Render a renderable

Implementation
Neos\Form\ViewHelpers\RenderRenderableViewHelper

Arguments

- `renderable` (NeosFormCoreModelRenderableRenderableInterface)

neos.form:renderValues

Renders the values of a form

Implementation
Neos\Form\ViewHelpers\RenderValuesViewHelper

Arguments

- `renderable` (NeosFormCoreModelRenderableRootRenderableInterface, *optional*): Relative Fusion path to be rendered
- `formRuntime` (NeosFormCoreRuntimeFormRuntime, *optional*): Relative Fusion path to be rendered
- `as` (string, *optional*): Relative Fusion path to be rendered

neos.form:translateElementProperty

ViewHelper to translate the property of a given form element based on its rendering options

Implementation
Neos\Form\ViewHelpers\TranslateElementPropertyViewHelper

Arguments

- `property` (string): The property to translate
- `element` (NeosFormCoreModelFormElementInterface, *optional*): Form element

1.4.4 Fusion ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

fusion:render

Render a Fusion object with a relative Fusion path, optionally pushing new variables onto the Fusion context.

Implementation
Neos\Fusion\ViewHelpers\RenderViewHelper

Arguments

- `path` (string): Relative Fusion path to be rendered
- `context` (array, *optional*): ReAdditional context variables to be set
- `fusionPackageKey` (string, *optional*): The key of the package to load Fusion from, if not from the current context
- `fusionFilePathPattern` (string, *optional*): Resource pattern to load Fusion from. Defaults to: resource://@package/Private/Fusion/

Examples

Simple:

```
Fusion:
some.given {
        path = Neos.Fusion:Template
        ...
}
ViewHelper:
<ts:render path="some.given.path" />
```

Expected result:

```
(the evaluated Fusion, depending on the given path)
```

Fusion from a foreign package:

```
<ts:render path="some.given.path" fusionPackageKey="Acme.Bookstore" />
```

Expected result:

```
(the evaluated Fusion, depending on the given path)
```

1.4.5 Media ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

neos.media:fileTypeIcon

Renders an <img> HTML tag for a file type icon for a given Neos.Media's asset instance

Implementation
Neos\Media\ViewHelpers\FileTypeIconViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `asset` (NeosMediaDomainModelAssetInterface, *optional*): An Asset object to determine the file type icon for. Alternatively $filename can be specified.
- `filename` (string, *optional*): A filename to determine the file type icon for. Alternatively $asset can be specified.
- `width` (integer, *optional*): Desired width of the icon
- `height` (integer, *optional*): Desired height of the icon

Examples

Rendering an asset file type icon:

```
<neos.media:fileTypeIcon asset="{assetObject}" height="16" />
```

Expected result:

```
(depending on the asset, no scaling applied)
<img src="_Resources/Static/Packages/Neos/Media/Icons/16px/jpg.png" height="16" alt=
↪"file type alt text" />
```

Rendering a file type icon by given filename:

```
<neos.media:fileTypeIcon filename="{someFilename}" height="16" />
```

Expected result:

```
(depending on the asset, no scaling applied)
<img src="_Resources/Static/Packages/Neos/Media/Icons/16px/jpg.png" height="16" alt=
↪"file type alt text" />
```

neos.media:form.checkbox

View Helper which creates a simple checkbox (<input type="checkbox">).

Implementation
Neos\Media\ViewHelpers\Form\CheckboxViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `name` (string, *optional*): Name of input tag
- `value` (mixed): Value of input tag. Required for checkboxes
- `property` (string, *optional*): Name of Object Property. If used in conjunction with <f:form object="...">, "name" and "value" properties will be ignored.
- `disabled` (string, *optional*): Specifies that the input element should be disabled when the page loads
- `errorClass` (string, *optional*): CSS class to set if there are errors for this view helper
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `checked` (boolean, *optional*): Specifies that the input element should be preselected
- `multiple` (boolean, *optional*): Specifies whether this checkbox belongs to a multivalue (is part of a checkbox group)

Examples

Example:

```
<neos.media:form.checkbox name="myCheckBox" value="someValue" />
```

Expected result:

```
<input type="checkbox" name="myCheckBox" value="someValue" />
```

Preselect:

```
<neos.media:form.checkbox name="myCheckBox" value="someValue" checked="{object.value} ==␣
↪5" />
```

Expected result:

```
<input type="checkbox" name="myCheckBox" value="someValue" checked="checked" />
(depending on $object)
```

Bind to object property:

```
<neos.media:form.checkbox property="interests" value="Neos" />
```

Expected result:

```
<input type="checkbox" name="user[interests][]" value="Neos" checked="checked" />
(depending on property "interests")
```

neos.media:format.relativeDate

Renders a DateTime formatted relative to the current date

Implementation
Neos\Media\ViewHelpers\Format\RelativeDateViewHelper

Arguments

- `date` (DateTimeInterface, *optional*): The date to be formatted

neos.media:image

Renders an <img> HTML tag from a given Neos.Media's image instance

Implementation
Neos\Media\ViewHelpers\ImageViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `alt` (string): Specifies an alternate text for an image
- `ismap` (string, *optional*): Specifies an image as a server-side image-map. Rarely used. Look at usemap instead
- `usemap` (string, *optional*): Specifies an image as a client-side image-map
- `loading` (string, *optional*): Specifies the loading attribute for an image
- `image` (NeosMediaDomainModelImageInterface, *optional*): The image to be rendered as an image
- `width` (integer, *optional*): Desired width of the image
- `maximumWidth` (integer, *optional*): Desired maximum width of the image
- `height` (integer, *optional*): Desired height of the image
- `maximumHeight` (integer, *optional*): Desired maximum height of the image
- `allowCropping` (boolean, *optional*): Whether the image should be cropped if the given sizes would hurt the aspect ratio
- `allowUpScaling` (boolean, *optional*): Whether the resulting image size might exceed the size of the original asset
- `async` (boolean, *optional*): Return asynchronous image URI in case the requested image does not exist already
- `preset` (string, *optional*): Preset used to determine image configuration
- `quality` (integer, *optional*): Quality of the image, from 0 to 100
- `format` (string, *optional*): Format for the image, jpg, jpeg, gif, png, wbmp, xbm, webp and bmp are supported

Examples

Rendering an image as-is:

```
<neos.media:image image="{imageObject}" alt="a sample image without scaling" />
```

Expected result:

```
(depending on the image, no scaling applied)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="120" height="180" alt="a sample␣
↪image without scaling" />
```

Rendering an image with scaling at a given width only:

```
<neos.media:image image="{imageObject}" maximumWidth="80" alt="sample" />
```

Expected result:

```
(depending on the image; scaled down to a maximum width of 80 pixels, keeping the aspect␣
↪ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="120" alt="sample" />
```

Rendering an image with scaling at given width and height, keeping aspect ratio:

```
<neos.media:image image="{imageObject}" maximumWidth="80" maximumHeight="80" alt="sample
↪" />
```

Expected result:

```
(depending on the image; scaled down to a maximum width and height of 80 pixels, keeping␣
↪the aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="53" height="80" alt="sample" />
```

Rendering an image with crop-scaling at given width and height:

```
<neos.media:image image="{imageObject}" maximumWidth="80" maximumHeight="80"␣
↪allowCropping="true" alt="sample" />
```

Expected result:

```
(depending on the image; scaled down to a width and height of 80 pixels, possibly␣
↪changing aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="80" alt="sample" />
```

Rendering an image with allowed up-scaling at given width and height:

```
<neos.media:image image="{imageObject}" maximumWidth="5000" allowUpScaling="true" alt=
↪"sample" />
```

Expected result:

```
(depending on the image; scaled up or down to a width 5000 pixels, keeping aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="80" alt="sample" />
```

neos.media:thumbnail

Renders an <img> HTML tag from a given Neos.Media's asset instance

Implementation
Neos\Media\ViewHelpers\ThumbnailViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `alt` (string): Specifies an alternate text for an asset
- `asset` (NeosMediaDomainModelAssetInterface): The asset to be rendered as a thumbnail
- `width` (integer, *optional*): Desired width of the thumbnail
- `maximumWidth` (integer, *optional*): Desired maximum width of the thumbnail
- `height` (integer, *optional*): Desired height of the thumbnail
- `maximumHeight` (integer, *optional*): Desired maximum height of the thumbnail
- `allowCropping` (boolean, *optional*): Whether the thumbnail should be cropped if the given sizes would hurt the aspect ratio
- `allowUpScaling` (boolean, *optional*): Whether the resulting thumbnail size might exceed the size of the original asset
- `async` (boolean, *optional*): Return asynchronous image URI in case the requested image does not exist already
- `preset` (string, *optional*): Preset used to determine image configuration
- `quality` (integer, *optional*): Quality of the image, from 0 to 100

Examples

Rendering an asset thumbnail:

```
<neos.media:thumbnail asset="{assetObject}" alt="a sample asset without scaling" />
```

Expected result:

```
(depending on the asset, no scaling applied)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="120" height="180" alt="a sample␣
↪asset without scaling" />
```

Rendering an asset thumbnail with scaling at a given width only:

```
<neos.media:thumbnail asset="{assetObject}" maximumWidth="80" alt="sample" />
```

Expected result:

```
(depending on the asset; scaled down to a maximum width of 80 pixels, keeping the aspect␣
↪ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="120" alt="sample" />
```

Rendering an asset thumbnail with scaling at given width and height, keeping aspect ratio:

```
<neos.media:thumbnail asset="{assetObject}" maximumWidth="80" maximumHeight="80" alt=
↪"sample" />
```

Expected result:

```
(depending on the asset; scaled down to a maximum width and height of 80 pixels, keeping␣
↪the aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="53" height="80" alt="sample" />
```

Rendering an asset thumbnail with crop-scaling at given width and height:

```
<neos.media:thumbnail asset="{assetObject}" maximumWidth="80" maximumHeight="80"␣
↪allowCropping="true" alt="sample" />
```

Expected result:

```
(depending on the asset; scaled down to a width and height of 80 pixels, possibly␣
↪changing aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="80" alt="sample" />
```

Rendering an asset thumbnail with allowed up-scaling at given width and height:

```
<neos.media:thumbnail asset="{assetObject}" maximumWidth="5000" allowUpScaling="true"␣
↪alt="sample" />
```

Expected result:

```
(depending on the asset; scaled up or down to a width 5000 pixels, keeping aspect ratio)
<img src="_Resources/Persistent/b29[...]95d.jpeg" width="80" height="80" alt="sample" />
```

neos.media:uri.image

Renders the src path of a thumbnail image of a given Neos.Media image instance

Implementation
Neos\Media\ViewHelpers\Uri\ImageViewHelper

Arguments

- `image` (NeosMediaDomainModelImageInterface, *optional*): The image to be rendered as an image
- `width` (integer, *optional*): Desired width of the image
- `maximumWidth` (integer, *optional*): Desired maximum width of the image
- `height` (integer, *optional*): Desired height of the image
- `maximumHeight` (integer, *optional*): Desired maximum height of the image
- `allowCropping` (boolean, *optional*): Whether the image should be cropped if the given sizes would hurt the aspect ratio
- `allowUpScaling` (boolean, *optional*): Whether the resulting image size might exceed the size of the original asset
- `async` (boolean, *optional*): Return asynchronous image URI in case the requested image does not exist already
- `preset` (string, *optional*): Preset used to determine image configuration
- `quality` (integer, *optional*): Quality of the image, from 0 to 100
- `format` (string, *optional*): Format for the image, jpg, jpeg, gif, png, wbmp, xbm, webp and bmp are supported

Examples

Rendering an image path as-is:

```
{neos.media:uri.image(image: imageObject)}
```

Expected result:

```
(depending on the image)
_Resources/Persistent/b29[...]95d.jpeg
```

Rendering an image path with scaling at a given width only:

```
{neos.media:uri.image(image: imageObject, maximumWidth: 80)}
```

Expected result:

```
(depending on the image; has scaled keeping the aspect ratio)
_Resources/Persistent/b29[...]95d.jpeg
```

neos.media:uri.thumbnail

Renders the src path of a thumbnail image of a given Neos.Media asset instance

Implementation
Neos\Media\ViewHelpers\Uri\ThumbnailViewHelper

Arguments

- `asset` (NeosMediaDomainModelAssetInterface): The asset to be rendered as a thumbnail
- `width` (integer, *optional*): Desired width of the thumbnail
- `maximumWidth` (integer, *optional*): Desired maximum width of the thumbnail
- `height` (integer, *optional*): Desired height of the thumbnail
- `maximumHeight` (integer, *optional*): Desired maximum height of the thumbnail
- `allowCropping` (boolean, *optional*): Whether the thumbnail should be cropped if the given sizes would hurt the aspect ratio
- `allowUpScaling` (boolean, *optional*): Whether the resulting thumbnail size might exceed the size of the original asset
- `async` (boolean, *optional*): Return asynchronous image URI in case the requested image does not exist already
- `preset` (string, *optional*): Preset used to determine image configuration
- `quality` (integer, *optional*): Quality of the image

Examples

Rendering an asset thumbnail path as-is:

```
{neos.media:uri.thumbnail(asset: assetObject)}
```

Expected result:

```
(depending on the asset)
_Resources/Persistent/b29[...]95d.jpeg
```

Rendering an asset thumbnail path with scaling at a given width only:

```
{neos.media:uri.thumbnail(asset: assetObject, maximumWidth: 80)}
```

Expected result:

```
(depending on the asset; has scaled keeping the aspect ratio)
_Resources/Persistent/b29[...]95d.jpeg
```

1.4.6 Neos ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

neos:backend.authenticationProviderLabel

Renders a label for the given authentication provider identifier

Implementation
Neos\Neos\ViewHelpers\Backend\AuthenticationProviderLabelViewHelper

Arguments

- `identifier` (string): The identifier to render the label for

neos:backend.changeStats

Displays a text-based "bar graph" giving an indication of the amount and type of changes done to something. Created for use in workspace management.

Implementation
Neos\Neos\ViewHelpers\Backend\ChangeStatsViewHelper

Arguments

- `changeCounts` (array): Expected keys: new, changed, removed

neos:backend.colorOfString

Generates a color code for a given string

Implementation
Neos\Neos\ViewHelpers\Backend\ColorOfStringViewHelper

Arguments

- `string` (string, *optional*): This is hashed (MD%) and then used as base for the resulting color, if not given the children are used
- `minimalBrightness` (integer, *optional*): Brightness, from 0 to 255

neos:backend.configurationCacheVersion

ViewHelper for rendering the current version identifier for the configuration cache.

Implementation
Neos\Neos\ViewHelpers\Backend\ConfigurationCacheVersionViewHelper

neos:backend.configurationTree

Render HTML markup for the full configuration tree in the Neos Administration -> Configuration Module.

For performance reasons, this is done inside a ViewHelper instead of Fluid itself.

Implementation
Neos\Neos\ViewHelpers\Backend\ConfigurationTreeViewHelper

Arguments

- `configuration` (array): Configuration to show

neos:backend.cssBuiltVersion

Returns a shortened md5 of the built CSS file

Implementation
Neos\Neos\ViewHelpers\Backend\CssBuiltVersionViewHelper

neos:backend.documentBreadcrumbPath

Render a bread crumb path by using the labels of documents leading to the given node path

Implementation
Neos\Neos\ViewHelpers\Backend\DocumentBreadcrumbPathViewHelper

Arguments

- `node` (NeosContentRepositoryDomainModelNodeInterface): Node

neos:backend.ifModuleAccessible

Condition ViewHelper that can evaluate whether the currently authenticated user can access a given Backend module

Note: This is a quick fix for https://github.com/neos/neos-development-collection/issues/2854 that will be obsolete once the whole Backend module logic is rewritten

Implementation
Neos\Neos\ViewHelpers\Backend\IfModuleAccessibleViewHelper

Arguments

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `condition` (boolean, *optional*): Condition expression conforming to Fluid boolean rules
- `modulePath` (string): Path of the module to evaluate
- `moduleConfiguration` (array): Configuration of the module to evaluate

neos:backend.interfaceLanguage

ViewHelper for rendering the current backend users interface language.

Implementation
Neos\Neos\ViewHelpers\Backend\InterfaceLanguageViewHelper

neos:backend.isAllowedToEditUser

Returns true, if the current user is allowed to edit the given user, false otherwise.

Implementation
Neos\Neos\ViewHelpers\Backend\IsAllowedToEditUserViewHelper

Arguments

- `user` (NeosNeosDomainModelUser): The user subject

neos:backend.javascriptConfiguration

ViewHelper for the backend JavaScript configuration. Renders the required JS snippet to configure the Neos backend.

Implementation
Neos\Neos\ViewHelpers\Backend\JavascriptConfigurationViewHelper

neos:backend.translate

Returns translated message using source message or key ID. uses the selected backend language * Also replaces all placeholders with formatted versions of provided values.

Implementation
Neos\Neos\ViewHelpers\Backend\TranslateViewHelper

Arguments

- `id` (string, *optional*): Id to use for finding translation (trans-unit id in XLIFF)
- `value` (string, *optional*): If $key is not specified or could not be resolved, this value is used. If this argument is not set, child nodes will be used to render the default
- `arguments` (array, *optional*): Numerically indexed array of values to be inserted into placeholders
- `source` (string, *optional*): Name of file with translations (use / as a directory separator)
- `package` (string, *optional*): Target package key. If not set, the current package key will be used
- `quantity` (mixed, *optional*): A number to find plural form for (float or int), NULL to not use plural forms
- `locale` (string, *optional*): An identifier of locale to use (NULL for use the default locale)

Examples

Translation by id:

```
<neos:backend.translate id="user.unregistered">Unregistered User</neos:backend.translate>
```

Expected result:

```
translation of label with the id "user.unregistered" and a fallback to "Unregistered User
↪"
```

Inline notation:

```
{neos:backend.translate(id: 'some.label.id', value: 'fallback result')}
```

Expected result:

```
translation of label with the id "some.label.id" and a fallback to "fallback result"
```

Custom source and locale:

```
<neos:backend.translate id="some.label.id" source="SomeLabelsCatalog" locale="de_DE"/>
```

Expected result:

```
translation from custom source "SomeLabelsCatalog" for locale "de_DE"
```

Custom source from other package:

```
<neos:backend.translate id="some.label.id" source="LabelsCatalog" package="OtherPackage"/
↪>
```

Expected result:

```
translation from custom source "LabelsCatalog" in "OtherPackage"
```

Arguments:

```
<neos:backend.translate arguments="{0: 'foo', 1: '99.9'}"><![CDATA[Untranslated {0} and
↪{1,number}]]></neos:backend.translate>
```

Expected result:

```
translation of the label "Untranslated foo and 99.9"
```

Translation by label:

```
<neos:backend.translate>Untranslated label</neos:backend.translate>
```

Expected result:

```
translation of the label "Untranslated label"
```

neos:backend.userInitials

Render user initials for a given username

This ViewHelper is *WORK IN PROGRESS* and *NOT STABLE YET*

Implementation
Neos\Neos\ViewHelpers\Backend\UserInitialsViewHelper

Arguments

- `format` (string, *optional*): Supported are "fullFirstName", "initials" and "fullName"

neos:backend.xliffCacheVersion

ViewHelper for rendering the current version identifier for the xliff cache.

Implementation
Neos\Neos\ViewHelpers\Backend\XliffCacheVersionViewHelper

neos:contentElement.editable

Renders a wrapper around the inner contents of the tag to enable frontend editing.

The wrapper contains the property name which should be made editable, and is by default a "div" tag. The tag to use can be given as *tag* argument to the ViewHelper.

In live workspace this just renders a tag with the specified $tag-name containing the value of the given $property. For logged in users with access to the Backend this also adds required attributes for the RTE to work.

Note: when passing a node you have to make sure a metadata wrapper is used around this that matches the given node (see contentElement.wrap - i.e. the WrapViewHelper).

Implementation
Neos\Neos\ViewHelpers\ContentElement\EditableViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `property` (string): Name of the property to render. Note: If this tag has child nodes, they overrule this argument!
- `tag` (string, *optional*): The name of the tag that should be wrapped around the property. By default this is a
- `node` (NeosContentRepositoryDomainModelNodeInterface, *optional*): The node of the content element. Optional, will be resolved from the Fusion context by default

neos:contentElement.wrap

A view helper for manually wrapping content editables.

Note that using this view helper is usually not necessary as Neos will automatically wrap editables of content elements.

By explicitly wrapping template parts with node meta data that is required for the backend to show properties in the inspector, this ViewHelper enables usage of the `contentElement.editable` ViewHelper outside of content element templates. This is useful if you want to make properties of a custom document node inline-editable.

Implementation
Neos\Neos\ViewHelpers\ContentElement\WrapViewHelper

Arguments

- `node` (NeosContentRepositoryDomainModelNodeInterface, *optional*): Node

neos:getType

View helper to check if a given value is an array.

Implementation
Neos\Neos\ViewHelpers\GetTypeViewHelper

Arguments

- `value` (mixed, *optional*): The value to get the type of

Examples

Basic usage:

```
{neos:getType(value: 'foo')}
```

Expected result:

```
string
```

Use with shorthand syntax:

```
{myValue -> neos:getType()}
```

Expected result:

```
string
(if myValue is a string)
```

neos:link.module

A view helper for creating links to modules.

Implementation
Neos\Neos\ViewHelpers\Link\ModuleViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element

- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor
- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document
- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `path` (string): Target module path
- `action` (string, *optional*): Target module action
- `arguments` (array, *optional*): Arguments
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (array, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true

Examples

Defaults:

```
<neos:link.module path="system/useradmin">some link</neos:link.module>
```

Expected result:

```
<a href="neos/system/useradmin">some link</a>
```

neos:link.node

A view helper for creating links with URIs pointing to nodes.

The target node can be provided as string or as a Node object; if not specified at all, the generated URI will refer to the current document node inside the Fusion context.

When specifying the `node` argument as string, the following conventions apply:

``node`` starts with `/`: The given path is an absolute node path and is treated as such. Example: `/sites/acmecom/home/about/us`

``node`` does not start with `/`: The given path is treated as a path relative to the current node. Examples: given that the current node is `/sites/acmecom/products/`, `stapler` results in `/sites/acmecom/products/stapler`, `../about` results in `/sites/acmecom/about/`, `./neos/info` results in `/sites/acmecom/products/neos/info`.

``node`` starts with a tilde character (``~``): The given path is treated as a path relative to the current site node. Example: given that the current node is `/sites/acmecom/products/`, `~/about/us` results in `/sites/acmecom/about/us`, `~` results in `/sites/acmecom`.

Implementation
Neos\Neos\ViewHelpers\Link\NodeViewHelper

Arguments

- `additionalAttributes` (array, *optional*): Additional tag attributes. They will be added directly to the resulting HTML tag.
- `data` (array, *optional*): Additional data-* attributes. They will each be added with a "data-" prefix.
- `class` (string, *optional*): CSS class(es) for this element
- `dir` (string, *optional*): Text direction for this HTML element. Allowed strings: "ltr" (left to right), "rtl" (right to left)
- `id` (string, *optional*): Unique (in this file) identifier for this HTML element.
- `lang` (string, *optional*): Language for this element. Use short names specified in RFC 1766
- `style` (string, *optional*): Individual CSS styles for this element
- `title` (string, *optional*): Tooltip text of element
- `accesskey` (string, *optional*): Keyboard shortcut to access this element
- `tabindex` (integer, *optional*): Specifies the tab order of this element
- `onclick` (string, *optional*): JavaScript evaluated for the onclick event
- `name` (string, *optional*): Specifies the name of an anchor
- `rel` (string, *optional*): Specifies the relationship between the current document and the linked document
- `rev` (string, *optional*): Specifies the relationship between the linked document and the current document
- `target` (string, *optional*): Specifies where to open the linked document
- `node` (mixed, *optional*): A node object, a string node path (absolute or relative), a string node://-uri or NULL
- `format` (string, *optional*): Format to use for the URL, for example "html" or "json"
- `absolute` (boolean, *optional*): If set, an absolute URI is rendered
- `arguments` (array, *optional*): Additional arguments to be passed to the UriBuilder (for example pagination parameters)
- `section` (string, *optional*): The anchor to be added to the URI
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `baseNodeName` (string, *optional*): The name of the base node inside the Fusion context to use for the Content-Context or resolving relative paths
- `nodeVariableName` (string, *optional*): The variable the node will be assigned to for the rendered child content
- `resolveShortcuts` (boolean, *optional*): DEPRECATED Parameter - ignored

Examples

Defaults:

```
<neos:link.node>some link</neos:link.node>
```

Expected result:

```
<a href="sites/mysite.com/homepage/about.html">some link</a>
(depending on current node, format etc.)
```

Generating a link with an absolute URI:

```
<neos:link.node absolute="{true}">bookmark this page</neos:link.node>
```

Expected result:

```
<a href="http://www.example.org/homepage/about.html">bookmark this page</a>
(depending on current workspace, current node, format, host etc.)
```

Target node given as absolute node path:

```
<neos:link.node node="/sites/exampleorg/contact/imprint">Corporate imprint</neos:link.
↪node>
```

Expected result:

```
<a href="contact/imprint.html">Corporate imprint</a>
(depending on current workspace, current node, format etc.)
```

Target node given as node://-uri:

```
<neos:link.node node="node://30e893c1-caef-0ca5-b53d-e5699bb8e506">Corporate imprint</
↪neos:link.node>
```

Expected result:

```
<a href="contact/imprint.html">Corporate imprint</a>
(depending on current workspace, current node, format etc.)
```

Target node given as relative node path:

```
<neos:link.node node="~/about/us">About us</neos:link.node>
```

Expected result:

```
<a href="about/us.html">About us</a>
(depending on current workspace, current node, format etc.)
```

Node label as tag content:

```
<neos:link.node node="/sites/exampleorg/contact/imprint" />
```

Expected result:

```
<a href="contact/imprint.html">Imprint</a>
(depending on current workspace, current node, format etc.)
```

Dynamic tag content involving the linked node's properties:

```
<neos:link.node node="about-us">see our <span>{linkedNode.label}</span> page</neos:link.
↪node>
```

Expected result:

```
<a href="about-us.html">see our <span>About Us</span> page</a>
(depending on current workspace, current node, format etc.)
```

neos:node.closestDocument

ViewHelper to find the closest document node to a given node

Implementation
Neos\Neos\ViewHelpers\Node\ClosestDocumentViewHelper

Arguments

- node (NeosContentRepositoryDomainModelNodeInterface): Node

neos:rendering.inBackend

ViewHelper to find out if Neos is rendering the backend.

Implementation
Neos\Neos\ViewHelpers\Rendering\InBackendViewHelper

Arguments

- node (NeosContentRepositoryDomainModelNodeInterface, *optional*): Node

Examples

Basic usage:

```
<f:if condition="{neos:rendering.inBackend()}">
  <f:then>
    Shown in the backend.
  </f:then>
  <f:else>
    Shown when not in backend.
  </f:else>
</f:if>
```

Expected result:

```
Shown in the backend.
```

neos:rendering.inEditMode

ViewHelper to find out if Neos is rendering an edit mode.

Implementation
Neos\Neos\ViewHelpers\Rendering\InEditModeViewHelper

Arguments

- node (NeosContentRepositoryDomainModelNodeInterface, *optional*): Optional Node to use context from
- mode (string, *optional*): Optional rendering mode name to check if this specific mode is active

Examples

Basic usage:

```
<f:if condition="{neos:rendering.inEditMode()}">
  <f:then>
    Shown for editing.
  </f:then>
  <f:else>
    Shown elsewhere (preview mode or not in backend).
  </f:else>
</f:if>
```

Expected result:

```
Shown for editing.
```

Advanced usage:

```
<f:if condition="{neos:rendering.inEditMode(mode: 'rawContent')}">
  <f:then>
    Shown just for rawContent editing mode.
  </f:then>
  <f:else>
    Shown in all other cases.
  </f:else>
</f:if>
```

Expected result:

```
Shown in all other cases.
```

neos:rendering.inPreviewMode

ViewHelper to find out if Neos is rendering a preview mode.

Implementation
Neos\Neos\ViewHelpers\Rendering\InPreviewModeViewHelper

Arguments

- node (NeosContentRepositoryDomainModelNodeInterface, *optional*): Optional Node to use context from
- mode (string, *optional*): Optional rendering mode name to check if this specific mode is active

Examples

Basic usage:

```
<f:if condition="{neos:rendering.inPreviewMode()}">
  <f:then>
    Shown in preview.
  </f:then>
  <f:else>
    Shown elsewhere (edit mode or not in backend).
  </f:else>
</f:if>
```

Expected result:

```
Shown in preview.
```

Advanced usage:

```
<f:if condition="{neos:rendering.inPreviewMode(mode: 'print')}">
  <f:then>
    Shown just for print preview mode.
  </f:then>
  <f:else>
    Shown in all other cases.
  </f:else>
</f:if>
```

Expected result:

```
Shown in all other cases.
```

neos:rendering.live

ViewHelper to find out if Neos is rendering the live website. Make sure you either give a node from the current context to the ViewHelper or have "node" set as template variable at least.

Implementation
Neos\Neos\ViewHelpers\Rendering\LiveViewHelper

Arguments

- `node` (NeosContentRepositoryDomainModelNodeInterface, *optional*): Node

Examples

Basic usage:

```
<f:if condition="{neos:rendering.live()}">
  <f:then>
    Shown outside the backend.
  </f:then>
  <f:else>
    Shown in the backend.
  </f:else>
</f:if>
```

Expected result:

```
Shown in the backend.
```

neos:standaloneView

A View Helper to render a fluid template based on the given template path and filename.

This will just set up a standalone Fluid view and render the template found at the given path and filename. Any arguments passed will be assigned to that template, the rendering result is returned.

Implementation
Neos\Neos\ViewHelpers\StandaloneViewViewHelper

Arguments

- `templatePathAndFilename` (string): Path and filename of the template to render
- `arguments` (array, *optional*): Arguments to assign to the template before rendering

Examples

Basic usage:

```
<neos:standaloneView templatePathAndFilename="fancyTemplatePathAndFilename" arguments="
↪{foo: bar, quux: baz}" />
```

Expected result:

```
<some><fancy/></html
(depending on template and arguments given)
```

neos:uri.module

A view helper for creating links to modules.

Implementation
Neos\Neos\ViewHelpers\Uri\ModuleViewHelper

Arguments

- `path` (string): Target module path
- `action` (string, *optional*): Target module action
- `arguments` (string, *optional*): Arguments
- `section` (string, *optional*): The anchor to be added to the URI
- `format` (string, *optional*): The requested format, e.g. ".html"
- `additionalParams` (string, *optional*): additional query parameters that won't be prefixed like $arguments (overrule $arguments)
- `addQueryString` (string, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (string, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true

Examples

Defaults:

```
<link rel="some-module" href="{neos:uri.module(path: 'system/useradmin')}" />
```

Expected result:

```
<link rel="some-module" href="neos/system/useradmin" />
```

neos:uri.node

A view helper for creating URIs pointing to nodes.

The target node can be provided as string or as a Node object; if not specified at all, the generated URI will refer to the current document node inside the Fusion context.

When specifying the `node` argument as string, the following conventions apply:

``node`` starts with `/`: The given path is an absolute node path and is treated as such. Example: `/sites/acmecom/home/about/us`

``node`` does not start with `/`: The given path is treated as a path relative to the current node. Examples: given that the current node is `/sites/acmecom/products/`, `stapler` results in `/sites/acmecom/products/stapler`, `../about` results in `/sites/acmecom/about/`, `./neos/info` results in `/sites/acmecom/products/neos/info`.

``node`` starts with a tilde character (`~`): The given path is treated as a path relative to the current site node. Example: given that the current node is `/sites/acmecom/products/`, `~/about/us` results in `/sites/acmecom/about/us`, `~` results in `/sites/acmecom`.

Implementation
Neos\Neos\ViewHelpers\Uri\NodeViewHelper

Arguments

- `node` (mixed, *optional*): A node object, a string node path (absolute or relative), a string node://-uri or NULL
- `format` (string, *optional*): Format to use for the URL, for example "html" or "json"
- `absolute` (boolean, *optional*): If set, an absolute URI is rendered
- `arguments` (array, *optional*): Additional arguments to be passed to the UriBuilder (for example pagination parameters)
- `section` (string, *optional*): The anchor to be added to the URI
- `addQueryString` (boolean, *optional*): If set, the current query parameters will be kept in the URI
- `argumentsToBeExcludedFromQueryString` (array, *optional*): arguments to be removed from the URI. Only active if $addQueryString = true
- `baseNodeName` (string, *optional*): The name of the base node inside the Fusion context to use for the ContentContext or resolving relative paths
- `resolveShortcuts` (boolean, *optional*): DEPRECATED Parameter - ignored

Examples

Default:

```
<neos:uri.node />
```

Expected result:

```
homepage/about.html
(depending on current workspace, current node, format etc.)
```

Generating an absolute URI:

```
<neos:uri.node absolute="{true"} />
```

Expected result:

```
http://www.example.org/homepage/about.html
(depending on current workspace, current node, format, host etc.)
```

Target node given as absolute node path:

```
<neos:uri.node node="/sites/acmecom/about/us" />
```

Expected result:

```
about/us.html
(depending on current workspace, current node, format etc.)
```

Target node given as relative node path:

```
<neos:uri.node node="~/about/us" />
```

Expected result:

```
about/us.html
(depending on current workspace, current node, format etc.)
```

Target node given as node://-uri:

```
<neos:uri.node node="node://30e893c1-caef-0ca5-b53d-e5699bb8e506" />
```

Expected result:

```
about/us.html
(depending on current workspace, current node, format etc.)
```

1.4.7 TYPO3 Fluid ViewHelper Reference

This reference was automatically generated from code on 2024-03-10

f:alias

Declares new variables which are aliases of other variables. Takes a "map"-Parameter which is an associative array which defines the shorthand mapping.

The variables are only declared inside the `<f:alias>...</f:alias>` tag. After the closing tag, all declared variables are removed again.

Using this ViewHelper can be a sign of weak architecture. If you end up using it extensively you might want to fine-tune your "view model" (the data you assign to the view).

Examples

Single alias

```
<f:alias map="{x: 'foo'}">{x}</f:alias>
```

Output:

```
foo
```

Multiple mappings

```
<f:alias map="{x: foo.bar.baz, y: foo.bar.baz.name}">
    {x.name} or {y}
</f:alias>
```

Output:

```
[name] or [name]
```

Depending on `{foo.bar.baz}`.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\AliasViewHelper

- `map` (array): Array that specifies which variables should be mapped to which alias

f:cache.disable

ViewHelper to disable template compiling

Inserting this ViewHelper at any point in the template, including inside conditions which do not get rendered, will forcibly disable the caching/compiling of the full template file to a PHP class.

Use this if for whatever reason your platform is unable to create or load PHP classes (for example on read-only file systems or when using an incompatible default cache backend).

Passes through anything you place inside the ViewHelper, so can safely be used as container tag, as self-closing or with inline syntax - all with the same result.

Examples

Self-closing

```
<f:cache.disable />
```

Inline mode

```
{f:cache.disable()}
```

Container tag

```
<f:cache.disable>
   Some output or Fluid code
</f:cache.disble>
```

Additional output is also not compilable because of the ViewHelper

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Cache\DisableViewHelper

f:cache.static

ViewHelper to force compiling to a static string

Used around chunks of template code where you want the output of said template code to be compiled to a static string (rather than a collection of compiled nodes, as is the usual behavior).

The effect is that none of the child ViewHelpers or nodes used inside this tag will be evaluated when rendering the template once it is compiled. It will essentially replace all logic inside the tag with a plain string output.

Works by turning the `compile` method into a method that renders the child nodes and returns the resulting content directly as a string variable.

You can use this with great effect to further optimise the performance of your templates: in use cases where chunks of template code depend on static variables (like thoese in `{settings}` for example) and those variables never change, and the template uses no other dynamic variables, forcing the template to compile that chunk to a static string can save a lot of operations when rendering the compiled template.

NB: NOT TO BE USED FOR CACHING ANYTHING OTHER THAN STRING- COMPATIBLE OUTPUT!

USE WITH CARE! WILL PRESERVE EVERYTHING RENDERED, INCLUDING POTENTIALLY SENSITIVE DATA CONTAINED IN OUTPUT!

Examples

Usage and effect

```
<f:if condition="{var}">Is always evaluated also when compiled</f:if>
<f:cache.static>
    <f:if condition="{othervar}">
        Will only be evaluated once and this output will be
        cached as a static string with no logic attached.
        The compiled template will not contain neither the
        condition ViewHelperNodes or the variable accessor
        that are used inside this node.
    </f:if>
</f:cache.static>
```

This is also evaluated when compiled (static node is closed):

```
<f:if condition="{var}">Also evaluated; is outside static node</f:if>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Cache\StaticViewHelper

f:cache.warmup

ViewHelper to insert variables which only apply during cache warmup and only apply if no other variables are specified for the warmup process.

If a chunk of template code is impossible to compile without additional variables, for example when rendering sections or partials using dynamic names, you can use this ViewHelper around that chunk and specify a set of variables which will be assigned only while compiling the template and only when this is done as part of cache warmup. The template chunk can then be compiled using those default variables.

This does not imply that only those variable values will be used by the compiled template. It only means that DEFAULT values of vital variables will be present during compiling.

If you find yourself completely unable to properly warm up a specific template file even with use of this ViewHelper, then you can consider using `f:cache.disable` ViewHelper to prevent the template compiler from even attempting to compile it.

USE WITH CARE! SOME EDGE CASES OF FOR EXAMPLE VIEWHELPERS WHICH REQUIRE SPECIAL VARIABLE TYPES MAY NOT BE SUPPORTED HERE DUE TO THE RUDIMENTARY NATURE OF VARIABLES YOU DEFINE.

Examples

Usage and effect

```
<f:cache.warmup variables="{foo: bar}">
   Template code depending on {foo} variable which is not
   assigned when warming up Fluid's caches. {foo} is only
   assigned if the variable does not already exist and the
   assignment only happens if Fluid is in warmup mode.
</f:cache.warmup>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Cache\WarmupViewHelper

- `variables` (array, *optional*): Array of variables to assign ONLY when compiling. See main class documentation.

f:case

Case ViewHelper that is only usable within the `f:switch` ViewHelper.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\CaseViewHelper

- `value` (mixed): Value to match in this case

f:comment

This ViewHelper prevents rendering of any content inside the tag.

Contents of the comment will still be **parsed** thus throwing an Exception if it contains syntax errors. You can put child nodes in CDATA tags to avoid this.

Using this ViewHelper won't have a notable effect on performance, especially once the template is parsed. However it can lead to reduced readability. You can use layouts and partials to split a large template into smaller parts. Using self-descriptive names for the partials can make comments redundant.

Examples

Commenting out fluid code

```
Before
<f:comment>
    This is completely hidden.
    <f:debug>This does not get rendered</f:debug>
</f:comment>
After
```

Output:

```
Before
After
```

Prevent parsing

```
<f:comment><![CDATA[
   <f:some.invalid.syntax />
]]></f:comment>
```

Output:

Will be nothing.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\CommentViewHelper

f:count

This ViewHelper counts elements of the specified array or countable object.

Examples

Count array elements

```
<f:count subject="{0:1, 1:2, 2:3, 3:4}" />
```

Output:

```
4
```

inline notation

```
{objects -> f:count()}
```

Output:

```
10 (depending on the number of items in ``{objects}``)
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\CountViewHelper

- `subject` (array, *optional*): Countable subject, array or Countable

f:cycle

This ViewHelper cycles through the specified values. This can be often used to specify CSS classes for example.

To achieve the "zebra class" effect in a loop you can also use the "iteration" argument of the **for** ViewHelper.

Examples

These examples could also be achieved using the "iteration" argument of the ForViewHelper.

Simple

```
<f:for each="{0:1, 1:2, 2:3, 3:4}" as="foo">
    <f:cycle values="{0: 'foo', 1: 'bar', 2: 'baz'}" as="cycle">
        {cycle}
    </f:cycle>
</f:for>
```

Output:

```
foobarbazfoo
```

Alternating CSS class

```
<ul>
    <f:for each="{0:1, 1:2, 2:3, 3:4}" as="foo">
        <f:cycle values="{0: 'odd', 1: 'even'}" as="zebraClass">
            <li class="{zebraClass}">{foo}</li>
        </f:cycle>
    </f:for>
</ul>
```

Output:

```
<ul>
    <li class="odd">1</li>
    <li class="even">2</li>
    <li class="odd">3</li>
    <li class="even">4</li>
</ul>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\CycleViewHelper

- `values` (array, *optional*): The array or object implementing ArrayAccess (for example SplObjectStorage) to iterated over
- `as` (strong): The name of the iteration variable

f:debug

This ViewHelper is only meant to be used during development.

Examples

Inline notation and custom title

```
{object -> f:debug(title: 'Custom title')}
```

Output:

```
all properties of {object} nicely highlighted (with custom title)
```

Only output the type

```
{object -> f:debug(typeOnly: true)}
```

Output:

```
the type or class name of {object}
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\DebugViewHelper

- `typeOnly` (boolean, *optional*): If TRUE, debugs only the type of variables
- `levels` (integer, *optional*): Levels to render when rendering nested objects/arrays
- `html` (boolean, *optional*): Render HTML. If FALSE, output is indented plaintext

f:defaultCase

A ViewHelper which specifies the "default" case when used within the `f:switch` ViewHelper.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\DefaultCaseViewHelper

f:else

Else-Branch of a condition. Only has an effect inside of `f:if`. See the `f:if` ViewHelper for documentation.

Examples

Output content if condition is not met

```
<f:if condition="{someCondition}">
    <f:else>
        condition was not true
    </f:else>
</f:if>
```

Output:

```
Everything inside the "else" tag is displayed if the condition evaluates to FALSE.
Otherwise nothing is outputted in this example.
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\ElseViewHelper

- `if` (boolean, *optional*): Condition expression conforming to Fluid boolean rules

f:for

Loop ViewHelper which can be used to iterate over arrays. Implements what a basic PHP `foreach()` does.

Examples

Simple Loop

```
<f:for each="{0:1, 1:2, 2:3, 3:4}" as="foo">{foo}</f:for>
```

Output:

```
1234
```

Output array key

```
<ul>
    <f:for each="{fruit1: 'apple', fruit2: 'pear', fruit3: 'banana', fruit4: 'cherry'}"
        as="fruit" key="label"
    >
        <li>{label}: {fruit}</li>
    </f:for>
</ul>
```

Output:

```
<ul>
    <li>fruit1: apple</li>
    <li>fruit2: pear</li>
    <li>fruit3: banana</li>
    <li>fruit4: cherry</li>
</ul>
```

Iteration information

```
<ul>
    <f:for each="{0:1, 1:2, 2:3, 3:4}" as="foo" iteration="fooIterator">
        <li>Index: {fooIterator.index} Cycle: {fooIterator.cycle} Total: {fooIterator.
↪total}{f:if(condition: fooIterator.isEven, then: ' Even')}{f:if(condition: fooIterator.
↪isOdd, then: ' Odd')}{f:if(condition: fooIterator.isFirst, then: ' First')}
↪{f:if(condition: fooIterator.isLast, then: ' Last')}</li>
    </f:for>
</ul>
```

Output:

```
<ul>
    <li>Index: 0 Cycle: 1 Total: 4 Odd First</li>
    <li>Index: 1 Cycle: 2 Total: 4 Even</li>
    <li>Index: 2 Cycle: 3 Total: 4 Odd</li>
    <li>Index: 3 Cycle: 4 Total: 4 Even Last</li>
</ul>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\ForViewHelper

- `each` (array): The array or SplObjectStorage to iterated over
- `as` (string): The name of the iteration variable
- `key` (string, *optional*): Variable to assign array key to
- `reverse` (boolean, *optional*): If TRUE, iterates in reverse
- `iteration` (string, *optional*): The name of the variable to store iteration information (index, cycle, isFirst, isLast, isEven, isOdd)

f:format.cdata

Outputs an argument/value without any escaping and wraps it with CDATA tags.

PAY SPECIAL ATTENTION TO SECURITY HERE (especially Cross Site Scripting), as the output is NOT SANITIZED!

Examples

Child nodes

```
<f:format.cdata>{string}</f:format.cdata>
```

Output:

```
<![CDATA[(Content of {string} without any conversion/escaping)]]>
```

Value attribute

```
<f:format.cdata value="{string}" />
```

Output:

```
<![CDATA[(Content of {string} without any conversion/escaping)]]>
```

Inline notation

```
{string -> f:format.cdata()}
```

Output:

```
<![CDATA[(Content of {string} without any conversion/escaping)]]>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Format\CdataViewHelper

- `value` (mixed, *optional*): The value to output

f:format.htmlspecialchars

Applies PHP `htmlspecialchars()` escaping to a value.

See http://www.php.net/manual/function.htmlspecialchars.php

Examples

Default notation

```
<f:format.htmlspecialchars>{text}</f:format.htmlspecialchars>
```

Output:

```
Text with & " ' < > * replaced by HTML entities (htmlspecialchars applied).
```

Inline notation

```
{text -> f:format.htmlspecialchars(encoding: 'ISO-8859-1')}
```

Output:

```
Text with & " ' < > * replaced by HTML entities (htmlspecialchars applied).
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Format\HtmlspecialcharsViewHelper

- `value` (string, *optional*): Value to format
- `keepQuotes` (boolean, *optional*): If TRUE quotes will not be replaced (ENT_NOQUOTES)
- `encoding` (string, *optional*): Encoding
- `doubleEncode` (boolean, *optional*): If FALSE html entities will not be encoded

f:format.printf

A ViewHelper for formatting values with printf. Either supply an array for the arguments or a single value.

See http://www.php.net/manual/en/function.sprintf.php

Examples

Scientific notation

```
<f:format.printf arguments="{number: 362525200}">%.3e</f:format.printf>
```

Output:

```
3.625e+8
```

Argument swapping

```
<f:format.printf arguments="{0: 3, 1: 'Kasper'}">%2$s is great, TYPO%1$d too. Yes, TYPO%1
↪$d is great and so is %2$s!</f:format.printf>
```

Output:

```
Kasper is great, TYPO3 too. Yes, TYPO3 is great and so is Kasper!
```

Single argument

```
<f:format.printf arguments="{1: 'TYPO3'}">We love %s</f:format.printf>
```

Output:

```
We love TYPO3
```

Inline notation

```
{someText -> f:format.printf(arguments: {1: 'TYPO3'})}
```

Output:

```
We love TYPO3
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Format\PrintfViewHelper

- `value` (string, *optional*): String to format
- `arguments` (array, *optional*): The arguments for vsprintf

f:format.raw

Outputs an argument/value without any escaping. Is normally used to output an ObjectAccessor which should not be escaped, but output as-is.

PAY SPECIAL ATTENTION TO SECURITY HERE (especially Cross Site Scripting), as the output is NOT SANITIZED!

Examples

Child nodes

```
<f:format.raw>{string}</f:format.raw>
```

Output:

```
(Content of ``{string}`` without any conversion/escaping)
```

Value attribute

```
<f:format.raw value="{string}" />
```

Output:

```
(Content of ``{string}`` without any conversion/escaping)
```

Inline notation

```
{string -> f:format.raw()}
```

Output:

```
(Content of ``{string}`` without any conversion/escaping)
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\Format\RawViewHelper

- `value` (mixed, *optional*): The value to output

f:groupedFor

Grouped loop ViewHelper. Loops through the specified values.

The groupBy argument also supports property paths.

Using this ViewHelper can be a sign of weak architecture. If you end up using it extensively you might want to fine-tune your "view model" (the data you assign to the view).

Examples

Simple

```
<f:groupedFor each="{0: {name: 'apple', color: 'green'}, 1: {name: 'cherry', color: 'red
↪'}, 2: {name: 'banana', color: 'yellow'}, 3: {name: 'strawberry', color: 'red'}}"
    as="fruitsOfThisColor" groupBy="color"
>
    <f:for each="{fruitsOfThisColor}" as="fruit">
        {fruit.name}
    </f:for>
</f:groupedFor>
```

Output:

```
apple cherry strawberry banana
```

Two dimensional list

```
<ul>
    <f:groupedFor each="{0: {name: 'apple', color: 'green'}, 1: {name: 'cherry', color:
↪'red'}, 2: {name: 'banana', color: 'yellow'}, 3: {name: 'strawberry', color: 'red'}}"␣
↪as="fruitsOfThisColor" groupBy="color" groupKey="color">
        <li>
            {color} fruits:
            <ul>
                <f:for each="{fruitsOfThisColor}" as="fruit" key="label">
                    <li>{label}: {fruit.name}</li>
                </f:for>
            </ul>
        </li>
    </f:groupedFor>
</ul>
```

Output:

```
<ul>
    <li>green fruits
        <ul>
            <li>0: apple</li>
        </ul>
    </li>
    <li>red fruits
        <ul>
            <li>1: cherry</li>
        </ul>
        <ul>
            <li>3: strawberry</li>
        </ul>
    </li>
    <li>yellow fruits
        <ul>
            <li>2: banana</li>
        </ul>
    </li>
</ul>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\GroupedForViewHelper

- `each` (array): The array or SplObjectStorage to iterated over
- `as` (string): The name of the iteration variable
- `groupBy` (string): Group by this property
- `groupKey` (string, *optional*): The name of the variable to store the current group

f:if

This ViewHelper implements an if/else condition.

Conditions:

As a condition is a boolean value, you can just use a boolean argument. Alternatively, you can write a boolean expression there. Boolean expressions have the following form:

XX Comparator YY

Comparator is one of: ==, !=, <, <=, >, >= and % The % operator converts the result of the % operation to boolean.

XX and YY can be one of:

- number
- Object Accessor
- Array
- a ViewHelper
- string

```
<f:if condition="{rank} > 100">
    Will be shown if rank is > 100
</f:if>
<f:if condition="{rank} % 2">
    Will be shown if rank % 2 != 0.
</f:if>
<f:if condition="{rank} == {k:bar()}">
    Checks if rank is equal to the result of the ViewHelper "k:bar"
</f:if>
<f:if condition="{foo.bar} == 'stringToCompare'">
    Will result in true if {foo.bar}'s represented value equals 'stringToCompare'.
</f:if>
```

Examples

Basic usage

```
<f:if condition="somecondition">
    This is being shown in case the condition matches
</f:if>
```

Output:

```
Everything inside the <f:if> tag is being displayed if the condition evaluates to TRUE.
```

If / then / else

```
<f:if condition="somecondition">
    <f:then>
        This is being shown in case the condition matches.
    </f:then>
    <f:else>
        This is being displayed in case the condition evaluates to FALSE.
    </f:else>
</f:if>
```

Output:

```
Everything inside the "then" tag is displayed if the condition evaluates to TRUE.
Otherwise, everything inside the "else"-tag is displayed.
```

inline notation

```
{f:if(condition: someCondition, then: 'condition is met', else: 'condition is not met')}
```

Output:

```
The value of the "then" attribute is displayed if the condition evaluates to TRUE.
Otherwise, everything the value of the "else"-attribute is displayed.
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\IfViewHelper

- `then` (mixed, *optional*): Value to be returned if the condition if met.
- `else` (mixed, *optional*): Value to be returned if the condition if not met.
- `condition` (boolean, *optional*): Condition expression conforming to Fluid boolean rules

f:inline

Inline Fluid rendering ViewHelper

Renders Fluid code stored in a variable, which you normally would have to render before assigning it to the view. Instead you can do the following (note, extremely simplified use case):

```
$view->assign('variable', 'value of my variable');
$view->assign('code', 'My variable: {variable}');
```

And in the template:

```
{code -> f:inline()}
```

Which outputs:

```
My variable: value of my variable
```

You can use this to pass smaller and dynamic pieces of Fluid code to templates, as an alternative to creating new partial templates.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\InlineViewHelper

- `code` (string, *optional*): Fluid code to be rendered as if it were part of the template rendering it. Can be passed as inline argument or tag content

f:layout

With this tag, you can select a layout to be used for the current template.

Examples

```
<f:layout name="main" />
```

Output:

```
(no output)
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\LayoutViewHelper

Arguments

- `name` (string, *optional*): Name of layout to use. If none given, "Default" is used.

f:or

If content is empty use alternative text

Implementation
TYPO3Fluid\Fluid\ViewHelpers\OrViewHelper

- `content` (mixed, *optional*): Content to check if empty
- `alternative` (mixed, *optional*): Alternative if content is empty
- `arguments` (array, *optional*): Arguments to be replaced in the resulting string, using sprintf

f:render

A ViewHelper to render a section, a partial, a specified section in a partial or a delegate ParsedTemplateInterface implementation.

Examples

Rendering partials

```
<f:render partial="SomePartial" arguments="{foo: someVariable}" />
```

Output:

```
the content of the partial "SomePartial". The content of the variable {someVariable}␣
↪will be available in the partial as {foo}
```

Rendering sections

```
<f:section name="someSection">This is a section. {foo}</f:section>
<f:render section="someSection" arguments="{foo: someVariable}" />
```

Output:

```
the content of the section "someSection". The content of the variable {someVariable}␣
↪will be available in the partial as {foo}
```

Rendering recursive sections

```
<f:section name="mySection">
    <ul>
        <f:for each="{myMenu}" as="menuItem">
            <li>
                {menuItem.text}
                <f:if condition="{menuItem.subItems}">
                    <f:render section="mySection" arguments="{myMenu: menuItem.subItems}
↪" />
                </f:if>
            </li>
        </f:for>
    </ul>
   </f:section>
   <f:render section="mySection" arguments="{myMenu: menu}" />
```

Output:

```
<ul>
    <li>menu1
        <ul>
          <li>menu1a</li>
          <li>menu1b</li>
        </ul>
    </li>
[...]
(depending on the value of {menu})
```

Passing all variables to a partial

```
<f:render partial="somePartial" arguments="{_all}" />
```

Output:

```
the content of the partial "somePartial".
Using the reserved keyword "_all", all available variables will be passed along to the␣
↪partial
```

Rendering via a delegate ParsedTemplateInterface implementation w/ custom arguments

```
<f:render delegate="My\Special\ParsedTemplateImplementation" arguments="{_all}" />
```

This will output whichever output was generated by calling `My\Special\ParsedTemplateImplementation->render()` with cloned RenderingContextInterface $renderingContext as only argument and content of arguments assigned in VariableProvider of cloned context. Supports all other input arguments including recursive rendering, contentAs argument, default value etc.

Note that while ParsedTemplateInterface supports returning a Layout name, this Layout will not be respected when rendering using this method. Only the `render()` method will be called!

Implementation
TYPO3Fluid\Fluid\ViewHelpers\RenderViewHelper

- `section` (string, *optional*): Section to render - combine with partial to render section in partial
- `partial` (string, *optional*): Partial to render, with or without section
- `delegate` (string, *optional*): Optional PHP class name of a permanent, included-in-app ParsedTemplateInterface implementation to override partial/section
- `renderable` (TYPO3FluidFluidCoreRenderingRenderableInterface, *optional*): Instance of a RenderableInterface implementation to be rendered
- `arguments` (array, *optional*): Array of variables to be transferred. Use {_all} for all variables
- `optional` (boolean, *optional*): If TRUE, considers the *section* optional. Partial never is.
- `default` (mixed, *optional*): Value (usually string) to be displayed if the section or partial does not exist
- `contentAs` (string, *optional*): If used, renders the child content and adds it as a template variable with this name for use in the partial/section

f:section

A ViewHelper to declare sections in templates for later use with e.g. the `f:render` ViewHelper.

Examples

Rendering sections

```
<f:section name="someSection">This is a section. {foo}</f:section>
<f:render section="someSection" arguments="{foo: someVariable}" />
```

Output:

```
the content of the section "someSection". The content of the variable {someVariable}␣
↪will be available in the partial as {foo}
```

Rendering recursive sections

```
<f:section name="mySection">
   <ul>
        <f:for each="{myMenu}" as="menuItem">
             <li>
               {menuItem.text}
               <f:if condition="{menuItem.subItems}">
                   <f:render section="mySection" arguments="{myMenu: menuItem.subItems}"␣
↪/>
               </f:if>
             </li>
        </f:for>
   </ul>
</f:section>
<f:render section="mySection" arguments="{myMenu: menu}" />
```

Output:

```
<ul>
    <li>menu1
        <ul>
            <li>menu1a</li>
            <li>menu1b</li>
        </ul>
    </li>
[...]
(depending on the value of {menu})
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\SectionViewHelper

- name (string): Name of the section

f:spaceless

Space Removal ViewHelper

Removes redundant spaces between HTML tags while preserving the whitespace that may be inside HTML tags. Trims the final result before output.

Heavily inspired by Twig's corresponding node type.

Usage of f:spaceless

```
<f:spaceless>
    <div>
        <div>
            <div>text

    text</div>
        </div>
    </div>
</f:spaceless>
```

Output:

```
<div><div><div>text

text</div></div></div>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\SpacelessViewHelper

f:switch

Switch ViewHelper which can be used to render content depending on a value or expression. Implements what a basic PHP `switch()` does.

An optional default case can be specified which is rendered if none of the `case` conditions matches.

Using this ViewHelper can be a sign of weak architecture. If you end up using it extensively you might want to consider restructuring your controllers/actions and/or use partials and sections. E.g. the above example could be achieved with **:html:`<f:render partial="title.{person.gender}" />`** and the partials "title.male.html", "title.female.html", ... Depending on the scenario this can be easier to extend and possibly contains less duplication.

Examples

Simple Switch statement

```
<f:switch expression="{person.gender}">
    <f:case value="male">Mr.</f:case>
    <f:case value="female">Mrs.</f:case>
    <f:defaultCase>Mr. / Mrs.</f:defaultCase>
</f:switch>
```

Output:

```
"Mr.", "Mrs." or "Mr. / Mrs." (depending on the value of {person.gender})
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\SwitchViewHelper

- `expression` (mixed): Expression to switch

f:then

`f:then` only has an effect inside of `f:if`. See the `f:if` ViewHelper for documentation.

Implementation
TYPO3Fluid\Fluid\ViewHelpers\ThenViewHelper

f:variable

Variable assigning ViewHelper

Assigns one template variable which will exist also after the ViewHelper is done rendering, i.e. adds template variables.

If you require a variable assignment which does not exist in the template after a piece of Fluid code is rendered, consider using `f:alias` ViewHelper instead.

Usages:

```
{f:variable(name: 'myvariable', value: 'some value')}
<f:variable name="myvariable">some value</f:variable>
{oldvariable -> f:format.htmlspecialchars() -> f:variable(name: 'newvariable')}
<f:variable name="myvariable"><f:format.htmlspecialchars>{oldvariable}</f:format.
↪htmlspecialchars></f:variable>
```

Implementation
TYPO3Fluid\Fluid\ViewHelpers\VariableViewHelper

- `value` (mixed, *optional*): Value to assign. If not in arguments then taken from tag content
- `name` (string): Name of variable to create

1.5 Fusion Reference

1.5.1 Neos.Fusion

This package contains general-purpose Fusion objects, which are usable both within Neos and standalone.

Neos.Fusion:Array

[key]
(string) A nested definition (simple value, expression or object) that evaluates to a string

[key].@ignoreProperties
(array) A list of properties to ignore from being "rendered" during evaluation

[key].@position
(string/integer) Define the ordering of the nested definition

---

Note: The Neos.Fusion:Array object has been renamed to Neos.Fusion:Join the old name is DEPRECATED;

Neos.Fusion:Join

Render multiple nested definitions and concatenate the results.

[key]
(string) A nested definition (simple value, expression or object) that evaluates to a string

[key].@ignoreProperties
(array) A list of properties to ignore from being "rendered" during evaluation

[key].@position
(string/integer) Define the ordering of the nested definition

@glue
(string) The glue used to join the items together (default = '').

The order in which nested definitions are evaluated are specified using their `@position` meta property. For this argument, the following sort order applies:

- `start [priority]` positions. The higher the priority, the earlier the object is added. If no priority is given, the element is sorted after all `start` elements with a priority.
- `[numeric ordering]` positions, ordered ascending.
- `end [priority]` positions. The higher the priority, the later the element is added. If no priority is given, the element is sorted before all `end` elements with a priority.

Furthermore, you can specify that an element should be inserted before or after a given other named element, using `before` and `after` syntax as follows:

- `before [namedElement] [optionalPriority]`: add this element before `namedElement`; the higher the priority the more in front of `namedElement` we will add it if multiple `before [namedElement]` statements exist. Statements without `[optionalPriority]` are added the farthest before the element.

  If `[namedElement]` does not exist, the element is added after all `start` positions.

- `after [namedElement] [optionalPriority]`: add this element after `namedElement`; the higher the priority the more closely after `namedElement` we will add it if multiple `after [namedElement]` statements exist. Statements without `[optionalPriority]` are added farthest after the element.

  If `[namedElement]` does not exist, the element is added before all all `end` positions.

Example Ordering:

```
# in this example, we would not need to use any @position property;
# as the default (document order) would then be used. However, the
# order (o1 ... o9) is *always* fixed, no matter in which order the
# individual statements are defined.

myString = Neos.Fusion:Join {
        o1 = Neos.NodeTypes:Text
        o1.@position = 'start 12'
        o2 = Neos.NodeTypes:Text
        o2.@position = 'start 5'
        o2 = Neos.NodeTypes:Text
        o2.@position = 'start'

        o3 = Neos.NodeTypes:Text
        o3.@position = '10'
        o4 = Neos.NodeTypes:Text
        o4.@position = '20'

        o5 = Neos.NodeTypes:Text
        o5.@position = 'before o6'

        o6 = Neos.NodeTypes:Text
        o6.@position = 'end'
        o7 = Neos.NodeTypes:Text
        o7.@position = 'end 20'
        o8 = Neos.NodeTypes:Text
        o8.@position = 'end 30'

        o9 = Neos.NodeTypes:Text
        o9.@position = 'after o8'
}
```

If no `@position` property is defined, the array key is used. However, we suggest to use `@position` and meaningful keys in your application, and not numeric ones.

Example of numeric keys (discouraged):

```
myString = Neos.Fusion:Join {
        10 = Neos.NodeTypes:Text
        20 = Neos.NodeTypes:Text
}
```

Neos.Fusion:Collection

Render each item in `collection` using `itemRenderer`.

collection
(array/Iterable, **required**) The array or iterable to iterate over

itemName
(string, defaults to `item`) Context variable name for each item

itemKey
(string, defaults to `itemKey`) Context variable name for each item key, when working with array

iterationName
(string, defaults to `iterator`) A context variable with iteration information will be available under the given name: `index` (zero-based), `cycle` (1-based), `isFirst`, `isLast`

itemRenderer
(string, **required**) The renderer definition (simple value, expression or object) will be called once for every collection element, and its results will be concatenated (if `itemRenderer` cannot be rendered the path `content` is used as fallback for convenience in afx)

Note: The Neos.Fusion:Collection object is DEPRECATED use Neos.Fusion:Loop instead.

Example using an object `itemRenderer`:

```
myCollection = Neos.Fusion:Collection {
        collection = ${[1, 2, 3]}
        itemName = 'element'
        itemRenderer = Neos.Fusion:Template {
                templatePath = 'resource://...'
                element = ${element}
        }
}
```

Example using an expression `itemRenderer`:

```
myCollection = Neos.Fusion:Collection {
        collection = ${[1, 2, 3]}
        itemName = 'element'
        itemRenderer = ${element * 2}
}
```

Neos.Fusion:RawCollection

Render each item in `collection` using `itemRenderer` and return the result as an array (opposed to *string* for *Neos.Fusion:Collection*)

collection
(array/Iterable, **required**) The array or iterable to iterate over

itemName
(string, defaults to `item`) Context variable name for each item

itemKey
(string, defaults to `itemKey`) Context variable name for each item key, when working with array

iterationName
(string, defaults to `iterator`) A context variable with iteration information will be available under the given name: `index` (zero-based), `cycle` (1-based), `isFirst`, `isLast`

itemRenderer
(mixed, **required**) The renderer definition (simple value, expression or object) will be called once for every collection element (if `itemRenderer` cannot be rendered the path `content` is used as fallback for convenience in afx)

Note: The Neos.Fusion:RawCollection object is DEPRECATED use Neos.Fusion:Map instead.**

Neos.Fusion:Loop

Render each item in `items` using `itemRenderer`.

items
(array/Iterable, **required**) The array or iterable to iterate over (to calculate `iterator.isLast` items have to be `countable`)

itemName
(string, defaults to `item`) Context variable name for each item

itemKey
(string, defaults to `itemKey`) Context variable name for each item key, when working with array

iterationName
(string, defaults to `iterator`) A context variable with iteration information will be available under the given name: `index` (zero-based), `cycle` (1-based), `isFirst`, `isLast`

itemRenderer
(string, **required**) The renderer definition (simple value, expression or object) will be called once for every collection element, and its results will be concatenated (if `itemRenderer` cannot be rendered the path `content` is used as fallback for convenience in afx)

@glue
(string) The glue used to join the items together (default = '').

Example using an object `itemRenderer`:

```
myLoop = Neos.Fusion:Loop {
        items = ${[1, 2, 3]}
        itemName = 'element'
        itemRenderer = Neos.Fusion:Template {
                templatePath = 'resource://...'
                element = ${element}
        }
}
```

Example using an expression `itemRenderer`:

```
myLoop = Neos.Fusion:Loop {
        items = ${[1, 2, 3]}
        itemName = 'element'
        itemRenderer = ${element * 2}
}
```

Neos.Fusion:Map

Render each item in `items` using `itemRenderer` and return the result as an array (opposed to *string* for *Neos.Fusion:Collection*)

items
(array/Iterable, **required**) The array or iterable to iterate over (to calculate `iterator.isLast` items have to be `countable`)

itemName
(string, defaults to `item`) Context variable name for each item

itemKey
(string, defaults to `itemKey`) Context variable name for each item key, when working with array

iterationName
(string, defaults to `iterator`) A context variable with iteration information will be available under the given name: `index` (zero-based), `cycle` (1-based), `isFirst`, `isLast`

itemRenderer
(mixed, **required**) The renderer definition (simple value, expression or object) will be called once for every collection element to render the item (if `itemRenderer` cannot be rendered the path `content` is used as fallback for convenience in afx)

keyRenderer
(mixed, **optional**) The renderer definition (simple value, expression or object) will be called once for every collection element to render the key in the result collection.

Neos.Fusion:Reduce

Reduce the given items to a single value by using `itemReducer`.

items
(array/Iterable, **required**) The array or iterable to iterate over (to calculate `iterator.isLast` items have to be `countable`)

itemName
(string, defaults to `item`) Context variable name for each item

itemKey
(string, defaults to `itemKey`) Context variable name for each item key, when working with array

carryName
(string, defaults to `carry`) Context variable that contains the result of the last iteration

iterationName
(string, defaults to `iterator`) A context variable with iteration information will be available under the given name: `index` (zero-based), `cycle` (1-based), `isFirst`, `isLast`

itemReducer
(mixed, **required**) The reducer definition (simple value, expression or object) that will be applied for every item.

initialValue
(mixed, defaults to `null`) The value that is passed to the first iteration or returned if the items are empty

Neos.Fusion:Case

Conditionally evaluate nested definitions.

Evaluates all nested definitions until the first `condition` evaluates to `TRUE`. The Case object will evaluate to a result using either `renderer`, `renderPath` or `type` on the matching definition.

[key]
A matcher definition

[key].condition
(boolean, **required**) A simple value, expression or object that will be used as a condition for this matcher

[key].type
(string) Object type to render (as string)

[key].element.*
(mixed) Properties for the rendered object (when using `type`)

[key].renderPath
(string) Relative or absolute path to render, overrules `type`

[key].renderer
(mixed) Rendering definition (simple value, expression or object), overrules `renderPath` and `type`

[key].@position
(string/integer) Define the ordering of the nested definition

Simple Example:

```
myCase = Neos.Fusion:Case {
        someCondition {
                condition = ${q(node).is('[instanceof MyNamespace:My.Special.SuperType]
↪')}
                type = 'MyNamespace:My.Special.Type'
        }

        otherCondition {
                @position = 'start'
                condition = ${q(documentNode).property('layout') == 'special'}
                renderer = ${'<marquee>' + q(node).property('content') + '</marquee>'}
        }

        fallback {
                condition = ${true}
                renderPath = '/myPath'
        }
}
```

The ordering of matcher definitions can be specified with the `@position` property (see *Neos.Fusion:Array*). Thus, the priority of existing matchers (e.g. the default Neos document rendering) can be changed by setting or overriding the `@position` property.

Note: The internal `Neos.Fusion:Matcher` object type is used to evaluate the matcher definitions which is based on the `Neos.Fusion:Renderer`.

Neos.Fusion:Renderer

The Renderer object will evaluate to a result using either `renderer`, `renderPath` or `type` from the configuration.

type
(string) Object type to render (as string)

element.*
(mixed) Properties for the rendered object (when using `type`)

renderPath
(string) Relative or absolute path to render, overrules `type`

renderer
(mixed) Rendering definition (simple value, expression or object), overrules `renderPath` and `type`

Simple Example:

```
myCase = Neos.Fusion:Renderer {
        type = 'Neos.Fusion:Value'
        element.value = 'hello World'
}
```

Note: This is especially handy if the prototype that should be rendered is determined via eel or passed via @context.

Neos.Fusion:Debug

Shows the result of Fusion Expressions directly.

title
(optional) Title for the debug output

plaintext
(boolean) If set true, the result will be shown as plaintext

[key]
(mixed) A nested definition (simple value, expression or object), `[key]` will be used as key for the resulting output

Example:

```
valueToDebug = "hello neos world"
valueToDebug.@process.debug = Neos.Fusion:Debug {
      title = 'Debug of hello world'

      # Additional values for debugging
      documentTitle = ${q(documentNode).property('title')}
      documentPath = ${documentNode.path}
}

# the initial value is not changed, so you can define the Debug prototype anywhere in␣
↪your Fusion code
```

Neos.Fusion:Debug.Console

Wraps the given value with a script tag to print it to the browser console. When used as process the script tag is appended to the processed value.

title
(optional) Title for the debug output

value
(mixed) The value to print to the console

method
(string, optional) The method to call on the browser console object

[key]
(mixed) Other arguments to pass to the console method

Example:

```
renderer.@process.debug = Neos.Fusion:Debug.Console {
  title = 'My props'
  value = ${props}
  method = 'table'
}
```

Multiple values:

```
renderer.@process.debug = Neos.Fusion:Debug.Console {
  value = ${props.foo}
  otherValue = ${props.other}
  thirdValue = ${props.third}
}
```

Color usage:

```
renderer.@process.debug = Neos.Fusion:Debug.Console {
  value = ${'%c' + node.identifier}
  color = 'color: red'
}
```

Neos.Fusion:Component

Create a component

[key]
(mixed) The public API of your component: Lazy evaluated props that will be available inside the current component's scope under the context `props` (is iterable)

@private.[key]
(mixed) Can only be set inside the root component declaration: Lazy evaluated private props that will be available inside the current component's scope under the context `private` (is not iterable / is only a proxy)

renderer
(mixed, **required**) The value which gets rendered

Note: The context `props` and `private` is only available in the components scope The component's scope will be available inside the *renderer* and *@private* and will extend inwards until inside another component's renderer That means inside *@private* it's even allowed to reference another private prop (be carefully of circular references, though!) But normal props are not inside the component's scope and thus cannot reference each other or `private`

Example:

```
prototype(Vendor.Site:Component) < prototype(Neos.Fusion:Component) {
    title = 'Hello World'
    titleTagName = 'h1'
    bold = false
```

(continues on next page)

(continued from previous page)

```
    @private {
        computedTitle = ${String.toLowercase(props.title)}
        funnyTitle = Neos.Fusion:Value {
            value = ${props.titleTagName + " " + private.computedTitle}
        }
    }

    renderer = Neos.Fusion:Tag {
        attributes.class {
            component = 'component'
            bold = ${props.bold && 'component--bold'}
        }
        content = Neos.Fusion:Join {
            headline = Neos.Fusion:Tag {
                tagName = ${props.titleTagName}
                content = ${private.funnyTitle}
            }
            // nestedComponentScope = Neos.Fusion:Component {
            //   prop1 = ${props.title} // works
            //   renderer = ${props.title} // doest work!
            // }
                    }
    }
}
```

Neos.Fusion:Fragment

A fragment is a component that renders the given *content* without additional markup. That way conditions can be defined for bigger chunks of afx instead of single tags.

content
(string) The value which gets rendered

Example:

```
renderer = afx`
        <Neos.Fusion:Fragment @if.isEnabled={props.enable}>
                <h1>Example</h1>
                <h2>Content</h2>
        </Neos.Fusion:Fragment>
`
```

Neos.Fusion:Augmenter

Modify given html content and add attributes. The augmenter can be used as processor or as a standalone prototype

content
(string) The content that shall be augmented

fallbackTagName
(string, defaults to `div`) If no single tag that can be augmented is found the content is wrapped into the fallback-tag before augmentation

[key]
All other fusion properties are added to the html content as html attributes

Example as a standalone augmenter:

```
augmentedContent = Neos.Fusion:Augmenter {

        content = Neos.Fusion:Join {
                title = Neos.Fusion:Tag {
                        @if.hasContent = ${this.content}
                        tagName = 'h2'
                        content = ${q(node).property('title')}
                }
                text = Neos.Fusion:Tag {
                        @if.hasContent = ${this.content}
                        tagName = 'p'
                        content = ${q(node).property('text')}
                }
        }

        fallbackTagName = 'header'

        class = 'header'
        data-foo = 'bar'
}
```

Example as a processor augmenter:

```
augmentedContent = Neos.Fusion:Tag {
        tagName = 'h2'
        content = 'Hello World'
        @process.augment = Neos.Fusion:Augmenter {
                        class = 'header'
                        data-foo = 'bar'
        }
}
```

Neos.Fusion:Template

Render a *Fluid template* specified by `templatePath`.

templatePath
(string, **required**) Path and filename for the template to be rendered, often a `resource://` URI

partialRootPath
(string) Path where partials are found on the file system

layoutRootPath
(string) Path where layouts are found on the file system

sectionName
(string) The Fluid <`f:section`> to be rendered, if given

[key]
(mixed) All remaining properties are directly passed into the Fluid template as template variables

Example:

```
myTemplate = Neos.Fusion:Template {
        templatePath = 'resource://My.Package/Private/Templates/FusionObjects/MyTemplate.
↪html'
        someDataAvailableInsideFluid = 'my data'
}

<div class="hero">
        {someDataAvailableInsideFluid}
</div>
```

Neos.Fusion:Value

Evaluate any value as a Fusion object

value
(mixed, **required**) The value to evaluate

Example:

```
myValue = Neos.Fusion:Value {
        value = 'Hello World'
}
```

Note: Most of the time this can be simplified by directly assigning the value instead of using the `Value` object.

Neos.Fusion:Match

Matches the given subject to a value

@subject
(string, **required**) The subject to match

@default
(mixed) The default to return when no match was found

[key]
(mixed) Definition list, the keys will be matched to the subject and their value returned.

Example:

```
myValue = Neos.Fusion:Match {
        @subject = 'hello'
        @default = 'World?'
        hello = 'Hello World'
        bye = 'Goodbye world'
}
```

Note: This can be used to simplify many usages of *Neos.Fusion:Case* when the subject is a string.

Neos.Fusion:Memo

Returns the result of previous calls with the same "discriminator"

discriminator
(string, **required**) Cache identifier

value
(mixed) The value to evaluate and store for future calls during rendering

Example:

```
prototype(My.Vendor:Expensive.Calculation) < prototype(Neos.Fusion:Memo) {
  discriminator = 'expensive-calculation'
  value = ${1+2}
}
```

Neos.Fusion:RawArray

Evaluate nested definitions as an array (opposed to *string* for *Neos.Fusion:Array*)

[key]
(mixed) A nested definition (simple value, expression or object), `[key]` will be used for the resulting array key

[key].@position
(string/integer) Define the ordering of the nested definition

Tip: For simple cases an expression with an array literal `${[1, 2, 3]}` might be easier to read

Note: The Neos.Fusion:RawArray object has been renamed to Neos.Fusion:DataStructure the old name is DEPRECATED;

Neos.Fusion:DataStructure

Evaluate nested definitions as an array (opposed to *string* for *Neos.Fusion:Array*)

[key]
(mixed) A nested definition (simple value, expression or object), `[key]` will be used for the resulting array key

[key].@position
(string/integer) Define the ordering of the nested definition

Tip: For simple cases an expression with an array literal `${[1, 2, 3]}` might be easier to read

Neos.Fusion:Tag

Render an HTML tag with attributes and optional body

tagName
(string) Tag name of the HTML element, defaults to `div`

omitClosingTag
(boolean) Whether to render the element `content` and the closing tag, defaults to `FALSE`

selfClosingTag
(boolean) Whether the tag is a self-closing tag with no closing tag. Will be resolved from `tagName` by default, so default HTML tags are treated correctly.

content
(string) The inner content of the element, will only be rendered if the tag is not self-closing and the closing tag is not omitted

attributes
(iterable) Tag attributes as key-value pairs. Default is `Neos.Fusion:DataStructure`. If a non iterable is returned the value is casted to string.

allowEmptyAttributes
(boolean) Whether empty attributes (HTML5 syntax) should be used for empty, false or null attribute values. By default this is `true`

Example:

```
htmlTag = Neos.Fusion:Tag {
        tagName = 'html'
        omitClosingTag = TRUE

        attributes {
                version = 'HTML+RDFa 1.1'
                xmlns = 'http://www.w3.org/1999/xhtml'
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Evaluates to:

```
<html version="HTML+RDFa 1.1" xmlns="http://www.w3.org/1999/xhtml">
```

Neos.Fusion:Attributes

A Fusion object to render HTML tag attributes. This object is used by the Neos_Fusion__Tag object to render the attributes of a tag. But it's also useful standalone to render extensible attributes in a Fluid template.

[key]
(string) A single attribute, array values are joined with whitespace. Boolean values will be rendered as an empty or absent attribute.

@allowEmpty
(boolean) Whether empty attributes (HTML5 syntax) should be used for empty, false or null attribute values

---

Note: The `Neos.Fusion:Attributes` object is DEPRECATED in favor of a solution inside Neos.Fusion:Tag which takes attributes as `Neos.Fusion:DataStructure` now. If you have to render attributes as string without a tag you can use `Neos.Fusion:Join` with ``@glue` but you will have to concatenate array attributes yourself.

Example:

```
attributes = Neos.Fusion:Attributes {
        foo = 'bar'
        class = Neos.Fusion:DataStructure {
                class1 = 'class1'
                class2 = 'class2'
        }
}
```

Evaluates to:

```
foo="bar" class="class1 class2"
```

Unsetting an attribute:

It's possible to unset an attribute by assigning `false` or `${null}` as a value. No attribute will be rendered for this case.

Neos.Fusion:Http.Message

A prototype based on *Neos.Fusion:Array* for rendering an HTTP message (response). It should be used to render documents since it generates a full HTTP response and allows to override the HTTP status code and headers.

httpResponseHead
(*Neos.Fusion:Http.ResponseHead*) An HTTP response head with properties to adjust the status and headers, the position in the `Array` defaults to the very beginning

[key]
(string) A nested definition (see *Neos.Fusion:Array*)

Example:

```
// Page extends from Http.Message
//
// prototype(Neos.Neos:Page) < prototype(Neos.Fusion:Http.Message)
//
page = Neos.Neos:Page {
        httpResponseHead.headers.Content-Type = 'application/json'
}
```

Neos.Fusion:Http.ResponseHead

A helper object to render the head of an HTTP response

statusCode
(integer) The HTTP status code for the response, defaults to `200`

headers.*
(string) An HTTP header that should be set on the response, the property name (e.g. `headers.Content-Type`) will be used for the header name

Neos.Fusion:ActionUri

Built a URI to a controller action

request
(ActionRequest, defaults to the the current `request`) The action request the uri is build from.

package
(string) The package key (e.g. `'My.Package'`)

subpackage
(string) The subpackage, empty by default

controller
(string) The controller name (e.g. `'Registration'`)

action
(string) The action name (e.g. `'new'`)

arguments
(array) Arguments to the action by named key

format
(string) An optional request format (e.g. `'html'`)

section
(string) An optional fragment (hash) for the URI

additionalParams
(array) Additional URI query parameters by named key

addQueryString
(boolean) Whether to keep the query parameters of the current URI

argumentsToBeExcludedFromQueryString
(array) Query parameters to exclude for `addQueryString`

absolute
(boolean) Whether to create an absolute URI

Example:

```
uri = Neos.Fusion:ActionUri {
        package = 'My.Package'
        controller = 'Registration'
        action = 'new'
}
```

A special case is generating URIs for links to Neos modules. In this case often the option *request = ${request.mainRequest}* is needed when linking to a controller outside of the context of the current subrequest.

Link to the content module:

```
uri = Neos.Fusion:ActionUri {
        request = ${request.mainRequest}
        package="Neos.Neos.Ui"
        controller="Backend"
        action = 'index'
        arguments.node = ${documentNode}
}
```

Link to backend modules (other than *content*):

```
uri = Neos.Fusion:ActionUri {
        request = ${request.mainRequest}
        action = "index"
        package = "Neos.Neos"
        controller = "Backend\\Module"
        arguments {
                module = 'administration/sites'
                moduleArguments {
                        @action = 'edit'
                        site = ${site}
                }
        }
}
```

Neos.Fusion:UriBuilder

Built a URI to a controller action

package
(string) The package key (e.g. `'My.Package'`)

subpackage
(string) The subpackage, empty by default

controller
(string) The controller name (e.g. `'Registration'`)

action
(string) The action name (e.g. `'new'`)

arguments
(array) Arguments to the action by named key

format
(string) An optional request format (e.g. `'html'`)

section
(string) An optional fragment (hash) for the URI

additionalParams
(array) Additional URI query parameters by named key

addQueryString
(boolean) Whether to keep the query parameters of the current URI

argumentsToBeExcludedFromQueryString
(array) Query parameters to exclude for `addQueryString`

absolute
(boolean) Whether to create an absolute URI

Note: The use of `Neos.Fusion:UriBuilder` is deprecated. Use _Neos_Fusion__ActionUri instead.

Example:

```
uri = Neos.Fusion:UriBuilder {
        package = 'My.Package'
        controller = 'Registration'
        action = 'new'
}
```

Neos.Fusion:ResourceUri

Build a URI to a static or persisted resource

path
(string) Path to resource, either a path relative to `Public` and `package` or a `resource://` URI

package
(string) The package key (e.g. `'My.Package'`)

resource
(Resource) A `Resource` object instead of `path` and `package`

localize
(boolean) Whether resource localization should be used, defaults to `true`

Example:

```
scriptInclude = Neos.Fusion:Tag {
        tagName = 'script'
        attributes {
                src = Neos.Fusion:ResourceUri {
                        path = 'resource://My.Package/Public/Scripts/App.js'
                }
        }
}
```

Neos.Fusion:Link.Action

Renders a link pointing to a controller/action

content
(string) content of the link tag

href
(string, default *Neos.Fusion:ActionUri*) The href for the link tag

[key]
(string) Other attributes for the link tag

Example:

```
link = Neos.Fusion:Link.Action {
        content = "register"
        class="action-link"
        href.package = 'My.Package'
        href.controller = 'Registration'
        href.action = 'new'
}

link = afx`
        <Neos.Fusion:Link.Action class="action-link" href.package="My.Package" href.
↪controller="Registration" href.action="new">
                register
        </Neos.Fusion:Link.Action>
`
```

Link to the content-module in afx:

```
<Neos.Fusion:Link.Action
                href.request={request.mainRequest}
                href.action="index"
                href.package="Neos.Neos.Ui"
                href.controller="Backend"
                href.arguments.node={node}
        >
                to content module
        </Neos.Fusion:Link.Action>
```

Link to backend-modules other than the content-module:

```
<Neos.Fusion:Link.Action
        href.request={request.mainRequest}
        href.action="index"
        href.package="Neos.Neos"
        href.controller="Backend\\Module"
        href.arguments.module='administration/sites'
        href.arguments.moduleArguments.@action='index'
>
        to site module
</Neos.Fusion:Link.Action>
```

Neos.Fusion:Link.Resource

Renders a link pointing to a resource

content
(string) content of the link tag

href
(string, default Neos_Fusion__ResouceUri) The href for the link tag

[key]
(string) Other attributes for the link tag

Example:

```
link = afx`
        <Neos.Fusion:Link.Resource class="resource-link" href.path="resource://Some.
↪Package/Public/Images/SomeImage.png">
                Some Link
        </Neos.Fusion:Link.Resource>
`
```

Neos.Fusion:CanRender

Check whether a Fusion prototype can be rendered. For being renderable a prototype must exist and have an implementation class, or inherit from an existing renderable prototype. The implementation class can be defined indirectly via base prototypes.

type
(string) The prototype name that is checked

path
(string) The fusion path name that is checked

Example:

```
canRender = Neos.Fusion:CanRender {
        type = 'My.Package:Prototype'
}
```

1.5.2 Neos.Neos Fusion Objects

The Fusion objects defined in the Neos package contain all Fusion objects which are needed to integrate a site. Often, it contains generic Fusion objects which do not need a particular node type to work on.

Neos.Neos:Page

Subclass of *Neos.Fusion:Http.Message*, which is based on *Neos.Fusion:Array*. Main entry point into rendering a page; responsible for rendering the `<html>` tag and everything inside.

doctype
(string) Defaults to `<!DOCTYPE html>`

htmlTag
(Neos_Fusion__Tag) The opening `<html>` tag

htmlTag.attributes
(*Neos.Fusion:Attributes*) Attributes for the `<html>` tag

headTag
(Neos_Fusion__Tag) The opening `<head>` tag

head
(*Neos.Fusion:Array*) HTML markup for the `<head>` tag

head.titleTag
(Neos_Fusion__Tag) The `<title>` tag

head.javascripts
(*Neos.Fusion:Array*) Script includes in the head should go here

head.stylesheets
(*Neos.Fusion:Array*) Link tags for stylesheets in the head should go here

body.templatePath
(string) Path to a fluid template for the page body

bodyTag
(Neos_Fusion__Tag) The opening `<body>` tag

bodyTag.attributes
(*Neos.Fusion:Attributes*) Attributes for the `<body>` tag

body
(*Neos.Fusion:Template*) HTML markup for the `<body>` tag

body.javascripts
(*Neos.Fusion:Array*) Body footer JavaScript includes

body.[key]
(mixed) Body template variables

Examples:

Rendering a simple page:

```
page = Page
page.body.templatePath = 'resource://My.Package/Private/MyTemplate.html'
// the following line is optional, but recommended for base CSS inclusions etc
page.body.sectionName = 'main'
```

Rendering content in the body:

Fusion:

```
page.body {
        sectionName = 'body'
        content.main = PrimaryContent {
                nodePath = 'main'
        }
}
```

Fluid:

```
<html>
        <body>
                <f:section name="body">
                        <div class="container">
                                {content.main -> f:format.raw()}
                        </div>
                </f:section>
        </body>
</html
```

Including stylesheets from a template section in the head:

```
page.head.stylesheets.mySite = Neos.Fusion:Template {
        templatePath = 'resource://My.Package/Private/MyTemplate.html'
        sectionName = 'stylesheets'
}
```

Adding body attributes with `bodyTag.attributes`:

```
page.bodyTag.attributes.class = 'body-css-class1 body-css-class2'
```

Neos.Neos:ContentCollection

Render nested content from a `ContentCollection` node. Individual nodes are rendered using the *Neos.Neos:ContentCase* object.

nodePath
(string, **required**) The relative node path of the `ContentCollection` (e.g. `'main'`)

@context.node
(Node) The content collection node, resolved from `nodePath` by default

tagName
(string) Tag name for the wrapper element

attributes
(*Neos.Fusion:Attributes*) Tag attributes for the wrapper element

Example:

```
page.body {
        content {
                main = Neos.Neos:PrimaryContent {
                        nodePath = 'main'
                }
                footer = Neos.Neos:ContentCollection {
                        nodePath = 'footer'
                }
        }
}
```

Neos.Neos:PrimaryContent

Primary content rendering, extends *Neos.Fusion:Case*. This is a prototype that can be used from packages to extend the default content rendering (e.g. to handle specific document node types).

nodePath
(string, **required**) The relative node path of the `ContentCollection` (e.g. `'main'`)

default
Default matcher that renders a ContentCollection

[key]
Additional matchers (see *Neos.Fusion:Case*)

Example for basic usage:

```
page.body {
        content {
                main = Neos.Neos:PrimaryContent {
                        nodePath = 'main'
                }
        }
}
```

Example for custom matcher:

```
prototype(Neos.Neos:PrimaryContent) {
        myArticle {
                condition = ${q(node).is('[instanceof My.Site:Article]')}
                renderer = My.Site:ArticleRenderer
        }
}
```

Neos.Neos:ContentCase

Render a content node, extends *Neos.Fusion:Case*. This is a prototype that is used by the default content rendering (*Neos.Neos:ContentCollection*) and can be extended to add custom matchers.

default
: Default matcher that renders a prototype of the same name as the node type name

[key]
: Additional matchers (see *Neos.Fusion:Case*)

Neos.Neos:Content

Base type to render content nodes, extends *Neos.Fusion:Template*. This prototype is extended by the auto-generated Fusion to define prototypes for each node type extending `Neos.Neos:Content`.

templatePath
: (string) The template path and filename, defaults to `'resource://[packageKey]/Private/Templates/NodeTypes/[nodeType].html'` (for auto-generated prototypes)

[key]
: (mixed) Template variables, all node type properties are available by default (for auto-generated prototypes)

attributes
: (*Neos.Fusion:Attributes*) Extensible attributes, used in the default templates

Example:

```
prototype(My.Package:MyContent) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://My.Package/Private/Templates/NodeTypes/MyContent.html'
        # Auto-generated for all node type properties
        # title = ${q(node).property('title')}
}
```

Neos.Neos:ContentComponent

Base type to render component based content-nodes, extends *Neos.Fusion:Component*. Features the same API as *Neos.Fusion:Component*, but it adds content element wrapping, so the node is correctly detected by the Neos.Ui

Neos.Neos:Editable

Create an editable tag for a property. In the frontend, only the content of the property gets rendered.

node
(node) A node instance that should be used to read the property. Default to *${node}*

property
(string) The name of the property which should be accessed

block
(boolean) Decides if the editable tag should be a block element (*div*) or an inline element (*span*). Default to *true*

Example:

```
title = Neos.Neos:Editable {
        property = 'title'
        block = false
}
```

Neos.Neos:Plugin

Base type to render plugin content nodes or static plugins. A *plugin* is a Flow controller that can implement arbitrary logic.

package
(string, **required**) The package key (e.g. *'My.Package'*)

subpackage
(string) The subpackage, defaults to empty

controller
(array) The controller name (e.g. 'Registration')

action
(string) The action name, defaults to *'index'*

argumentNamespace
(string) Namespace for action arguments, will be resolved from node type by default

[key]
(mixed) Pass an internal argument to the controller action (access with argument name `__key`)

Example:

```
prototype(My.Site:Registration) < prototype(Neos.Neos:Plugin) {
        package = 'My.Site'
        controller = 'Registration'
}
```

Example with argument passed to controller action:

```
prototype(My.Site:Registration) < prototype(Neos.Neos:Plugin) {
  package = 'My.Site'
  controller = 'Registration'
  action = 'register'
```

(continues on next page)

(continued from previous page)

```
  additionalArgument = 'foo'
}
```

Get argument in controller action:

```
public function registerAction()
{
  $additionalArgument = $this->request->getInternalArgument('__additionalArgument');
  [...]
}
```

Neos.Neos:Menu

Render a menu with items for nodes. Extends *Neos.Fusion:Template*.

templatePath
(string) Override the template path

entryLevel
(integer) Start the menu at the given depth

maximumLevels
(integer) Restrict the maximum depth of items in the menu (relative to `entryLevel`)

startingPoint
(Node) The parent node of the first menu level (defaults to `node` context variable)

lastLevel
(integer) Restrict the menu depth by node depth (relative to site node)

filter
(string) Filter items by node type (e.g. `'!My.Site:News,Neos.Neos:Document'`), defaults to `'Neos.Neos:Document'`

renderHiddenInIndex
(boolean) Whether nodes with `hiddenInIndex` should be rendered, defaults to `false`

itemCollection
(array) Explicitly set the Node items for the menu (alternative to `startingPoints` and levels)

attributes
(*Neos.Fusion:Attributes*) Extensible attributes for the whole menu

normal.attributes
(*Neos.Fusion:Attributes*) Attributes for normal state

active.attributes
(*Neos.Fusion:Attributes*) Attributes for active state

current.attributes
(*Neos.Fusion:Attributes*) Attributes for current state

Note: The `items` of the `Menu` are internally calculated with the prototype *Neos.Neos:MenuItems* which you can use directly aswell.

Menu item properties:

node
(Node) A node instance (with resolved shortcuts) that should be used to link to the item

originalNode
(Node) Original node for the item

state
(string) Menu state of the item: `'normal'`, `'current'` (the current node) or `'active'` (ancestor of current node)

label
(string) Full label of the node

menuLevel
(integer) Menu level the item is rendered on

Examples:

Custom menu template:

```
menu = Neos.Neos:Menu {
        entryLevel = 1
        maximumLevels = 3
        templatePath = 'resource://My.Site/Private/Templates/FusionObjects/MyMenu.html'
}
```

Menu including site node:

```
menu = Neos.Neos:Menu {
        itemCollection = ${q(site).add(q(site).children('[instanceof Neos.Neos:Document]
↪')).get()}
}
```

Menu with custom starting point:

```
menu = Neos.Neos:Menu {
        entryLevel = 2
        maximumLevels = 1
        startingPoint = ${q(site).children('[uriPathSegment="metamenu"]').get(0)}
}
```

Neos.Neos:BreadcrumbMenu

Render a breadcrumb (ancestor documents), based on *Neos.Neos:Menu*.

Example:

```
breadcrumb = Neos.Neos:BreadcrumbMenu
```

Note: The `items` of the `BreadcrumbMenu` are internally calculated with the prototype *Neos.Neos:MenuItems* which you can use directly aswell.

Neos.Neos:DimensionsMenu

Create links to other node variants (e.g. variants of the current node in other dimensions) by using this Fusion object.

If the `dimension` setting is given, the menu will only include items for this dimension, with all other configured dimension being set to the value(s) of the current node. Without any `dimension` being configured, all possible variants will be included.

If no node variant exists for the preset combination, a `NULL` node will be included in the item with a state `absent`.

dimension
(optional, string): name of the dimension which this menu should be based on. Example: "language".

presets
(optional, array): If set, the presets rendered will be taken from this list of preset identifiers

includeAllPresets
(boolean, default **false**) If TRUE, include all presets, not only allowed combinations

renderHiddenInIndex
(boolean, default **true**) If TRUE, render nodes which are marked as "hidded-in-index"

In the template for the menu, each `item` has the following properties:

node
(Node) A node instance (with resolved shortcuts) that should be used to link to the item

state
(string) Menu state of the item: `normal`, `current` (the current node), `absent`

label
(string) Label of the item (the dimension preset label)

menuLevel
(integer) Menu level the item is rendered on

dimensions
(array) Dimension values of the node, indexed by dimension name

targetDimensions
(array) The target dimensions, indexed by dimension name and values being arrays with `value`, `label` and `isPinnedDimension`

Note: The `DimensionMenu` is an alias to `DimensionsMenu`, available for compatibility reasons only.

Note: The `items` of the `DimensionsMenu` are internally calculated with the prototype *Neos.Neos:DimensionsMenuItems* which you can use directly aswell.

Examples

Minimal Example, outputting a menu with all configured dimension combinations:

```
variantMenu = Neos.Neos:DimensionsMenu
```

This example will create two menus, one for the 'language' and one for the 'country' dimension:

```
languageMenu = Neos.Neos:DimensionsMenu {
        dimension = 'language'
}
countryMenu = Neos.Neos:DimensionsMenu {
        dimension = 'country'
}
```

If you only want to render a subset of the available presets or manually define a specific order for a menu, you can override the "presets":

```
languageMenu = Neos.Neos:DimensionsMenu {
        dimension = 'language'
        presets = ${['en_US', 'de_DE']} # no matter how many languages are defined, only␣
↪these two are displayed.
}
```

In some cases, it can be good to ignore the availability of variants when rendering a dimensions menu. Consider a situation with two independent menus for country and language, where the following variants of a node exist (language / country):

- english / Germany
- german / Germany
- english / UK

If the user selects UK, only english will be linked in the language selector. German is only available again, if the user switches back to Germany first. This can be changed by setting the `includeAllPresets` option:

```
languageMenu = Neos.Neos:DimensionsMenu {
        dimension = 'language'
        includeAllPresets = true
}
```

Now the language menu will try to find nodes for all languages, if needed the menu items will point to a different country than currently selected. The menu tries to find a node to link to by using the current preset for the language (in this example) and the default presets for any other dimensions. So if fallback rules are in place and a node can be found, it is used.

Note: The `item.targetDimensions` will contain the "intended" dimensions, so that information can be used to inform the user about the potentially unexpected change of dimensions when following such a link.

Only if the current node is not available at all (even after considering default presets with their fallback rules), no node be assigned (so no link will be created and the items will have the `absent` state.)

Neos.Neos:MenuItems

Create a list of menu-items items for nodes.

entryLevel
(integer) Start the menu at the given depth

maximumLevels
(integer) Restrict the maximum depth of items in the menu (relative to `entryLevel`)

startingPoint
(Node) The parent node of the first menu level (defaults to `node` context variable)

lastLevel
(integer) Restrict the menu depth by node depth (relative to site node)

filter
(string) Filter items by node type (e.g. `'!My.Site:News,Neos.Neos:Document'`), defaults to `'Neos.Neos:Document'`

renderHiddenInIndex
(boolean) Whether nodes with `hiddenInIndex` should be rendered, defaults to `false`

itemCollection
(array) Explicitly set the Node items for the menu (alternative to `startingPoints` and levels)

itemUriRenderer
(*Neos.Neos:NodeUri*) prototype to use for rendering the URI of each item

MenuItems item properties:

node
(Node) A node instance (with resolved shortcuts) that should be used to link to the item

originalNode
(Node) Original node for the item

state
(string) Menu state of the item: `'normal'`, `'current'` (the current node) or `'active'` (ancestor of current node)

label
(string) Full label of the node

menuLevel
(integer) Menu level the item is rendered on

uri
(string) Frontend URI of the node

Examples:

```
menuItems = Neos.Neos:MenuItems {
        entryLevel = 1
        maximumLevels = 3
}
```

MenuItems including site node:

```
menuItems = Neos.Neos:MenuItems {
        itemCollection = ${q(site).add(q(site).children('[instanceof Neos.Neos:Document]
↪')).get()}
}
```

Menu with custom starting point:

```
menuItems = Neos.Neos:MenuItems {
        entryLevel = 2
        maximumLevels = 1
        startingPoint = ${q(site).children('[uriPathSegment="metamenu"]').get(0)}
}
```

Menu with absolute uris:

```
menuItems = Neos.Neos:MenuItems {
        itemUriRenderer {
                absolute = true
        }
}
```

Neos.Neos:BreadcrumbMenuItems

Create a list of of menu-items for a breadcrumb (ancestor documents), based on *Neos.Neos:MenuItems*.

Example:

```
breadcrumbItems = Neos.Neos:BreadcrumbMenuItems
```

Neos.Neos:DimensionsMenuItems

Create a list of menu-items for other node variants (e.g. variants of the current node in other dimensions) by using this Fusion object.

If the `dimension` setting is given, the menu will only include items for this dimension, with all other configured dimension being set to the value(s) of the current node. Without any `dimension` being configured, all possible variants will be included.

If no node variant exists for the preset combination, a `NULL` node will be included in the item with a state `absent`.

dimension
(optional, string): name of the dimension which this menu should be based on. Example: "language".

presets
(optional, array): If set, the presets rendered will be taken from this list of preset identifiers

includeAllPresets
(boolean, default **false**) If TRUE, include all presets, not only allowed combinations

renderHiddenInIndex
(boolean, default **true**) If TRUE, render nodes which are marked as "hidded-in-index"

Each `item` has the following properties:

node
(Node) A node instance (with resolved shortcuts) that should be used to link to the item

state
(string) Menu state of the item: `normal`, `current` (the current node), `absent`

label
(string) Label of the item (the dimension preset label)

menuLevel
(integer) Menu level the item is rendered on

dimensions
(array) Dimension values of the node, indexed by dimension name

targetDimensions
(array) The target dimensions, indexed by dimension name and values being arrays with `value`, `label` and `isPinnedDimension`

Examples

Minimal Example, outputting a menu with all configured dimension combinations:

```
variantMenuItems = Neos.Neos:DimensionsMenuItems
```

This example will create two menus, one for the 'language' and one for the 'country' dimension:

```
languageMenuItems = Neos.Neos:DimensionsMenuItems {
        dimension = 'language'
}
countryMenuItems = Neos.Neos:DimensionsMenuItems {
        dimension = 'country'
}
```

If you only want to render a subset of the available presets or manually define a specific order for a menu, you can override the "presets":

```
languageMenuItems = Neos.Neos:DimensionsMenuItems {
        dimension = 'language'
        presets = ${['en_US', 'de_DE']} # no matter how many languages are defined, only␣
↪these two are displayed.
}
```

In some cases, it can be good to ignore the availability of variants when rendering a dimensions menu. Consider a situation with two independent menus for country and language, where the following variants of a node exist (language / country):

- english / Germany
- german / Germany
- english / UK

If the user selects UK, only english will be linked in the language selector. German is only available again, if the user switches back to Germany first. This can be changed by setting the `includeAllPresets` option:

```
languageMenuItems = Neos.Neos:DimensionsMenuItems {
        dimension = 'language'
        includeAllPresets = true
}
```

Now the language menu will try to find nodes for all languages, if needed the menu items will point to a different country than currently selected. The menu tries to find a node to link to by using the current preset for the language (in this example) and the default presets for any other dimensions. So if fallback rules are in place and a node can be found, it is used.

Note: The `item.targetDimensions` will contain the "intended" dimensions, so that information can be used to inform the user about the potentially unexpected change of dimensions when following such a link.

Only if the current node is not available at all (even after considering default presets with their fallback rules), no node be assigned (so no link will be created and the items will have the `absent` state.)

Neos.Neos:NodeUri

Build a URI to a node. Accepts the same arguments as the node link/uri view helpers.

node
: (string/Node) A node object or a node path (relative or absolute) or empty to resolve the current document node

format
: (string) An optional request format (e.g. `'html'`)

section
: (string) An optional fragment (hash) for the URI

additionalParams
: (array) Additional URI query parameters.

argumentsToBeExcludedFromQueryString
: (array) Query parameters to exclude for `addQueryString`

addQueryString
(boolean) Whether to keep current query parameters, defaults to `FALSE`

absolute
(boolean) Whether to create an absolute URI, defaults to `FALSE`

baseNodeName
(string) Base node context variable name (for relative paths), defaults to `'documentNode'`

Example:

```
nodeLink = Neos.Neos:NodeUri {
        node = ${q(node).parent().get(0)}
}
```

Neos.Neos:NodeLink

Renders an anchor tag pointing to the node given via the argument. Based on *Neos.Neos:NodeUri*. The link text is the node label, unless overridden.

*****
All *Neos.Neos:NodeUri* properties

attributes
(*Neos.Fusion:Attributes*) Link tag attributes

content
(string) The label of the link, defaults to `node.label`.

Example:

```
nodeLink = Neos.Neos:NodeLink {
        node = ${q(node).parent().get(0)}
}
```

Note: By default no `title` is generated. By setting `attributes.title = ${node.label}` the label is rendered as title.

Neos.Neos:ImageUri

Get a URI to a (thumbnail) image for an asset.

asset
(Asset) An asset object (`Image`, `ImageInterface` or other `AssetInterface`)

width
(integer) Desired width of the image

maximumWidth
(integer) Desired maximum height of the image

height
(integer) Desired height of the image

maximumHeight
(integer) Desired maximum width of the image

allowCropping
(boolean) Whether the image should be cropped if the given sizes would hurt the aspect ratio, defaults to `FALSE`

allowUpScaling
(boolean) Whether the resulting image size might exceed the size of the original image, defaults to `FALSE`

async
(boolean) Return asynchronous image URI in case the requested image does not exist already, defaults to `FALSE`

quality
(integer) Image quality, from 0 to 100

format
(string) Format for the image, jpg, jpeg, gif, png, wbmp, xbm, webp and bmp are supported

preset
(string) Preset used to determine image configuration, if set all other resize attributes will be ignored

Example:

```
logoUri = Neos.Neos:ImageUri {
        asset = ${q(node).property('image')}
        width = 100
        height = 100
        allowCropping = TRUE
        allowUpScaling = TRUE
}
```

Neos.Neos:ImageTag

Render an image tag for an asset.

*
All *Neos.Neos:ImageUri* properties

attributes
(*Neos.Fusion:Attributes*) Image tag attributes

Per default, the attribute loading is set to `'lazy'`. To fetch a resource immediately, you can set `attributes.loading` to `null`, `false` or `'eager'`.

Example:

```
logoImage = Neos.Neos:ImageTag {
        asset = ${q(node).property('image')}
        maximumWidth = 400
        attributes.alt = 'A company logo'
}
```

Neos.Neos:ConvertUris

Convert internal node and asset URIs (`node://...` or `asset://...`) in a string to public URIs and allows for overriding the target attribute for external links and resource links.

value
(string) The string value, defaults to the `value` context variable to work as a processor by default

node
(Node) The current node as a reference, defaults to the `node` context variable

externalLinkTarget
(string) Override the target attribute for external links, defaults to `_blank`. Can be disabled with an empty value.

resourceLinkTarget
(string) Override the target attribute for resource links, defaults to `_blank`. Can be disabled with an empty value.

forceConversion
(boolean) Whether to convert URIs in a non-live workspace, defaults to `FALSE`

absolute
(boolean) Can be used to convert node URIs to absolute links, defaults to `FALSE`

setNoOpener
(boolean) Sets the rel="noopener" attribute to external links, which is good practice, defaults to `TRUE`

setExternal
(boolean) Sets the rel="external" attribute to external links. Defaults to `TRUE`

Example:

```
prototype(My.Site:Special.Type) {
        title.@process.convertUris = Neos.Neos:ConvertUris
}
```

Neos.Neos:ContentElementWrapping

Processor to augment rendered HTML code with node metadata that allows the Neos UI to select the node and show node properties in the inspector. This is especially useful if your renderer prototype is not derived from `Neos.Neos:Content`.

The processor expects being applied on HTML code with a single container tag that is augmented.

node
(Node) The node of the content element. Optional, will use the Fusion context variable `node` by default.

Example:

```
prototype(Vendor.Site:ExampleContent) {
        value = '<div>Example</div>'

        # The following line must not be removed as it adds required meta data
        # to edit content elements in the backend
        @process.contentElementWrapping = Neos.Neos:ContentElementWrapping {
                @position = 'end'
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Neos.Neos:ContentElementEditable

Processor to augment an HTML tag with metadata for inline editing to make a rendered representation of a property editable.

The processor expects beeing applied to an HTML tag with the content of the edited property.

node
: (Node) The node of the content element. Optional, will use the Fusion context variable `node` by default.

property
: (string) Node property that should be editable

Example:

```
renderer = Neos.Fusion:Tag {
        tagName = 'h1'
        content = ${q(node).property('title')}
        @process.contentElementEditableWrapping = Neos.Neos:ContentElementEditable {
                property = 'title'
        }
}
```

1.6 Eel Helpers Reference

This reference was automatically generated from code on 2024-03-10

1.6.1 Api

Implemented in: `Neos\Neos\Ui\Fusion\Helper\ApiHelper`

Api.emptyArrayToObject(array)

Converts an empty array to an empty object. Does nothing if array is not empty.

Use this helper to prevent associative arrays from being converted to non-associative arrays by json_encode. This is an internal helper and might change without further notice FIXME: Probably better to produce objects in the first place "upstream".

- `array` (array) Associative array which may be empty

Return (array|stdClass) Non-empty associative array or empty object

1.6.2 Array

Array helpers for Eel contexts

The implementation uses the JavaScript specificiation where applicable, including EcmaScript 6 proposals.

See https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Array for a documentation and specification of the JavaScript implementation.

Implemented in: `Neos\Eel\Helper\ArrayHelper`

Array.concat(array1, array2, array_)

Concatenate arrays or values to a new array

- `array1` (iterable|mixed) First array or value
- `array2` (iterable|mixed) Second array or value
- `array_` (iterable|mixed, *optional*) Optional variable list of additional arrays / values

Return (array) The array with concatenated arrays or values

Array.every(array, callback)

Check if all elements in an array pass a test given by the calback, passing each element and key as arguments

Example:

```
Array.every([1, 2, 3, 4], x => x % 2 == 0) // == false
Array.every([2, 4, 6, 8], x => x % 2) // == true
```

- `array` (iterable) Array of elements to test
- `callback` (callable) Callback for testing elements, current value and key will be passed as arguments

Return (bool) True if all elements passed the test

Array.filter(array, callback)

Filter an array by a test given as the callback, passing each element and key as arguments

Examples:

Array.filter([1, 2, 3, 4], x => x % 2 == 0) // == [2, 4] Array.filter([‘foo’, ‘bar’, ‘baz’], (x, index) => index < 2) // == [‘foo’, ‘bar’]

- `array` (iterable) Array of elements to filter
- `callback` (callable, *optional*) Callback for testing if an element should be included in the result, current value and key will be passed as arguments

Return (array) The array with elements where callback returned true

Array.first(array)

Get the first element of an array

- `array` (iterable) The array

Return (mixed)

Array.flip(array)

Exchanges all keys with their associated values in an array

Note that the values of array need to be valid keys, i.e. they need to be either int or string. If a value has several occurrences, the latest key will be used as its value, and all others will be lost.

- `array` (iterable)

Return (array) The array with flipped keys and values

Array.indexOf(array, searchElement, fromIndex)

Returns the first index at which a given element can be found in the array, or -1 if it is not present

- `array` (iterable) The array
- `searchElement` (mixed) The element value to find
- `fromIndex` (int, *optional*) Position in the array to start the search.

Return (int)

Array.isEmpty(array)

Check if an array is empty

- `array` (iterable) The array

Return (bool) true if the array is empty

Array.join(array, separator)

Join values of an array with a separator

- `array` (iterable) Array with values to join
- `separator` (string, *optional*) A separator for the values

Return (string) A string with the joined values separated by the separator

Array.keys(array)

Get the array keys

- `array` (iterable) The array

Return (array)

Array.ksort(array)

Sort an array by key

- `array` (iterable) The array to sort

Return (array) The sorted array

Array.last(array)

Get the last element of an array

- `array` (iterable) The array

Return (mixed)

Array.length(array)

Get the length of an array

- `array` (iterable) The array

Return (int)

Array.map(array, callback)

Apply the callback to each element of the array, passing each element and key as arguments

Examples:

```
Array.map([1, 2, 3, 4], x => x * x)
Array.map([1, 2, 3, 4], (x, index) => x * index)
```

- `array` (iterable) Array of elements to map
- `callback` (callable) Callback to apply for each element, current value and key will be passed as arguments

Return (array) The array with callback applied, keys will be preserved

Array.pop(array)

Removes the last element from an array

Note: This differs from the JavaScript behavior of Array.pop which will return the popped element.

An empty array will result in an empty array again.

- `array` (iterable)

Return (array) The array without the last element

Array.push(array, element)

Insert one or more elements at the end of an array

Allows to push multiple elements at once:

```
Array.push(array, e1, e2)
```

- `array` (iterable|scalar|null)
- `element` (mixed)

Return (array) The array with the inserted elements

Array.random(array)

Picks a random element from the array

- `array` (array)

Return (mixed) A random entry or null if the array is empty

Array.range(start, end, step)

Create an array containing a range of elements

If a step value is given, it will be used as the increment between elements in the sequence. step should be given as a positive number. If not specified, step will default to 1.

- `start` (mixed) First value of the sequence.
- `end` (mixed) The sequence is ended upon reaching the end value.
- `step` (int, *optional*) The increment between items, will default to 1.

Return (array) Array of elements from start to end, inclusive.

Array.reduce(array, callback, initialValue)

Apply the callback to each element of the array and accumulate a single value

Examples:

```
Array.reduce([1, 2, 3, 4], (accumulator, currentValue) => accumulator + currentValue) //␣
↪== 10
Array.reduce([1, 2, 3, 4], (accumulator, currentValue) => accumulator + currentValue, 1)␣
↪// == 11
```

- `array` (iterable) Array of elements to reduce to a value
- `callback` (callable) Callback for accumulating values, accumulator, current value and key will be passed as arguments
- `initialValue` (mixed, *optional*) Initial value, defaults to first item in array and callback starts with second entry

Return (mixed)

Array.reverse(array)

Returns an array in reverse order

- `array` (iterable) The array

Return (array)

Array.set(array, key, value)

Set the specified key in the the array

- `array` (iterable)
- `key` (string|integer) the key that should be set
- `value` (mixed) the value to assign to the key

Return (array) The modified array.

Array.shift(array)

Remove the first element of an array

Note: This differs from the JavaScript behavior of Array.shift which will return the shifted element.

An empty array will result in an empty array again.

- `array` (iterable)

Return (array) The array without the first element

Array.shuffle(array, preserveKeys)

Shuffle an array

Randomizes entries an array with the option to preserve the existing keys. When this option is set to false, all keys will be replaced

- `array` (iterable)
- `preserveKeys` (bool, *optional*) Wether to preserve the keys when shuffling the array

Return (array) The shuffled array

Array.slice(array, begin, end)

Extract a portion of an indexed array

- `array` (iterable) The array (with numeric indices)
- `begin` (int)
- `end` (int, *optional*)

Return (array)

Array.some(array, callback)

Check if at least one element in an array passes a test given by the calback, passing each element and key as arguments

Example:

```
Array.some([1, 2, 3, 4], x => x % 2 == 0) // == true
Array.some([1, 2, 3, 4], x => x > 4) // == false
```

- `array` (iterable) Array of elements to test
- `callback` (callable) Callback for testing elements, current value and key will be passed as arguments

Return (bool) True if at least one element passed the test

Array.sort(array)

Sorts an array

The sorting is done first by numbers, then by characters.

Internally natsort() is used as it most closely resembles javascript's sort(). Because there are no real associative arrays in Javascript, keys of the array will be preserved.

- `array` (iterable)

Return (array) The sorted array

Array.splice(array, offset, length, replacements)

Replaces a range of an array by the given replacements

Allows to give multiple replacements at once:

```
Array.splice(array, 3, 2, 'a', 'b')
```

- `array` (iterable)
- `offset` (int) Index of the first element to remove
- `length` (int, *optional*) Number of elements to remove
- `replacements` (mixed, *optional*) Elements to insert instead of the removed range

Return (array) The array with removed and replaced elements

Array.unique(array)

Removes duplicate values from an array

- `array` (iterable) The input array

Return (array) The filtered array.

Array.unshift(array, element)

Insert one or more elements at the beginning of an array

Allows to insert multiple elements at once:

```
Array.unshift(array, e1, e2)
```

- `array` (iterable)
- `element` (mixed)

Return (array) The array with the inserted elements

Array.values(array)

Get the array values

- `array` (iterable) The array

Return (array)

1.6.3 BaseUri

This is a purely internal helper to provide baseUris for Caching. It will be moved to a more sensible package in the future so do not rely on the classname for now.

Implemented in: `Neos\Fusion\Eel\BaseUriHelper`

BaseUri.getConfiguredBaseUriOrFallbackToCurrentRequest(fallbackRequest)

- `fallbackRequest` (ServerRequestInterface|null, *optional*)

Return (UriInterface)

1.6.4 Configuration

Configuration helpers for Eel contexts

Implemented in: `Neos\Eel\Helper\ConfigurationHelper`

Configuration.setting(settingPath)

Return the specified settings

Examples:

```
Configuration.setting('Neos.Flow.core.context') == 'Production'

Configuration.setting('Acme.Demo.speedMode') == 'light speed'
```

- `settingPath` (string)

Return (mixed)

1.6.5 ContentDimensions

Implemented in: `Neos\Neos\Ui\Fusion\Helper\ContentDimensionsHelper`

ContentDimensions.allowedPresetsByName(dimensions)

- `dimensions` (array) Dimension values indexed by dimension name

Return (array) Allowed preset names for the given dimension combination indexed by dimension name

ContentDimensions.contentDimensionsByName()

Return (array) Dimensions indexed by name with presets indexed by name

1.6.6 Date

Date helpers for Eel contexts

Implemented in: `Neos\Eel\Helper\DateHelper`

Date.add(date, interval)

Add an interval to a date and return a new DateTime object

- `date` (DateTime)
- `interval` (string|DateInterval)

Return (DateTime)

Date.create(time)

Get a date object by given date or time format

Examples:

```
Date.create('2018-12-04')
Date.create('first day of next year')
```

- `time` (String) A date/time string. For valid formats see http://php.net/manual/en/datetime.formats.php

Return (DateTime)

Date.dayOfMonth(dateTime)

Get the day of month of a date

- `dateTime` (DateTimeInterface)

Return (integer) The day of month of the given date

Date.diff(dateA, dateB)

Get the difference between two dates as a DateInterval object

- `dateA` (DateTime)
- `dateB` (DateTime)

Return (DateInterval)

Date.format(date, format)

Format a date (or interval) to a string with a given format

See formatting options as in PHP date()

- `date` (integer|string|DateTime|DateInterval)
- `format` (string)

Return (string)

Date.formatCldr(date, cldrFormat, locale)

Format a date to a string with a given cldr format

- `date` (integer|string|DateTime)
- `cldrFormat` (string) Format string in CLDR format (see http://cldr.unicode.org/translation/date-time)
- `locale` (null|string, *optional*) String locale - example (de|en|ru_RU)

Return (string)

Date.hour(dateTime)

Get the hour of a date (24 hour format)

- `dateTime` (DateTimeInterface)

Return (integer) The hour of the given date

Date.minute(dateTime)

Get the minute of a date

- `dateTime` (DateTimeInterface)

Return (integer) The minute of the given date

Date.month(dateTime)

Get the month of a date

- `dateTime` (DateTimeInterface)

Return (integer) The month of the given date

Date.now()

Get the current date and time

Examples:

```
Date.now().timestamp
```

Return (DateTime)

Date.parse(string, format)

Parse a date from string with a format to a DateTime object

- `string` (string)
- `format` (string)

Return (DateTime)

Date.second(dateTime)

Get the second of a date

- `dateTime` (DateTimeInterface)

Return (integer) The second of the given date

Date.subtract(date, interval)

Subtract an interval from a date and return a new DateTime object

- `date` (DateTime)
- `interval` (string|DateInterval)

Return (DateTime)

Date.today()

Get the current date

Return (DateTime)

Date.year(dateTime)

Get the year of a date

- `dateTime` (DateTimeInterface)

Return (integer) The year of the given date

1.6.7 File

Helper to read files.

Implemented in: `Neos\Eel\Helper\FileHelper`

File.exists(filepath)

Check if the given file path exists

- `filepath` (string)

Return (bool)

File.fileInfo(filepath)

Get file name and path information

- `filepath` (string)

Return (array) with keys dirname, basename, extension (if any), and filename

File.getSha1(filepath)

- `filepath` (string)

Return (string)

File.readFile(filepath)

Read and return the files contents for further use.

- `filepath` (string)

Return (string)

File.stat(filepath)

Get file information like creation and modification times as well as size.

- `filepath` (string)

Return (array) with keys mode, uid, gid, size, atime, mtime, ctime, (blksize, blocks, dev, ino, nlink, rdev)

1.6.8 Form.Schema

Implemented in: `Neos\Fusion\Form\Runtime\Helper\SchemaHelper`

Form.Schema.array()

Create an array schema.

Return (SchemaInterface)

Form.Schema.arrayOf(schema)

Create a date schema for an array by providing a schema for all items

- `schema` (SchemaInterface) The schema for the items of the array

Return (SchemaInterface)

Form.Schema.boolean()

Create a boolean schema

Return (SchemaInterface)

Form.Schema.date(format)

Create a date schema. The php value will be DateTime

- `format` (string, *optional*) The format default is "Y-m-d

Return (SchemaInterface)

Form.Schema.float()

Create a float schema

Return (SchemaInterface)

Form.Schema.forType(type)

Create a schema for the given type

- `type` (string) The type or className that is expected

Return (SchemaInterface)

Form.Schema.integer()

Create a integer schema

Return (SchemaInterface)

Form.Schema.resource(collection)

Create a resource schema

- `collection` (string, *optional*) The collection new resources are put into

Return (SchemaInterface)

Form.Schema.string()

Create a string schema

Return (SchemaInterface)

1.6.9 Json

JSON helpers for Eel contexts

Implemented in: `Neos\Eel\Helper\JsonHelper`

Json.parse(json, associativeArrays)

JSON decode the given string

- `json` (string)
- `associativeArrays` (boolean, *optional*)

Return (mixed)

Json.stringify(value, options)

JSON encode the given value

Usage example for options:

Json.stringify(value, ['JSON_UNESCAPED_UNICODE', 'JSON_FORCE_OBJECT'])

- `value` (mixed)
- `options` (array, *optional*) Array of option constant names as strings

Return (string)

1.6.10 Math

Math helpers for Eel contexts

The implementation sticks to the JavaScript specificiation including EcmaScript 6 proposals.

See https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Math for a documentation and specification of the JavaScript implementation.

Implemented in: `Neos\Eel\Helper\MathHelper`

Math.abs(x)

- **x** (float, *optional*) A number

Return (float) The absolute value of the given value

Math.acos(x)

- **x** (float) A number

Return (float) The arccosine (in radians) of the given value

Math.acosh(x)

- **x** (float) A number

Return (float) The hyperbolic arccosine (in radians) of the given value

Math.asin(x)

- **x** (float) A number

Return (float) The arcsine (in radians) of the given value

Math.asinh(x)

- **x** (float) A number

Return (float) The hyperbolic arcsine (in radians) of the given value

Math.atan(x)

- **x** (float) A number

Return (float) The arctangent (in radians) of the given value

Math.atan2(y, x)

- **y** (float) A number
- **x** (float) A number

Return (float) The arctangent of the quotient of its arguments

Math.atanh(x)

- **x** (float) A number

Return (float) The hyperbolic arctangent (in radians) of the given value

Math.cbrt(x)

- **x** (float) A number

Return (float) The cube root of the given value

Math.ceil(x)

- **x** (float) A number

Return (float) The smallest integer greater than or equal to the given value

Math.cos(x)

- **x** (float) A number given in radians

Return (float) The cosine of the given value

Math.cosh(x)

- **x** (float) A number

Return (float) The hyperbolic cosine of the given value

Math.exp(x)

- **x** (float) A number

Return (float) The power of the Euler's constant with the given value (e^x)

Math.expm1(x)

- **x** (float) A number

Return (float) The power of the Euler's constant with the given value minus 1 (e^x - 1)

Math.floor(x)

- **x** (float) A number

Return (float) The largest integer less than or equal to the given value

Math.getE()

Return (float) Euler's constant and the base of natural logarithms, approximately 2.718

Math.getLN10()

Return (float) Natural logarithm of 10, approximately 2.303

Math.getLN2()

Return (float) Natural logarithm of 2, approximately 0.693

Math.getLOG10E()

Return (float) Base 10 logarithm of E, approximately 0.434

Math.getLOG2E()

Return (float) Base 2 logarithm of E, approximately 1.443

Math.getPI()

Return (float) Ratio of the circumference of a circle to its diameter, approximately 3.14159

Math.getSQRT1_2()

Return (float) Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707

Math.getSQRT2()

Return (float) Square root of 2, approximately 1.414

Math.hypot(x, y, z_)

- `x` (float) A number
- `y` (float) A number
- `z_` (float, *optional*) Optional variable list of additional numbers

Return (float) The square root of the sum of squares of the arguments

Math.isFinite(x)

Test if the given value is a finite number

This is equivalent to the global isFinite() function in JavaScript.

- **x** (mixed) A value

Return (boolean) true if the value is a finite (not NAN) number

Math.isInfinite(x)

Test if the given value is an infinite number (INF or -INF)

This function has no direct equivalent in JavaScript.

- **x** (mixed) A value

Return (boolean) true if the value is INF or -INF

Math.isNaN(x)

Test if the given value is not a number (either not numeric or NAN)

This is equivalent to the global isNaN() function in JavaScript.

- **x** (mixed) A value

Return (boolean) true if the value is not a number

Math.log(x)

- **x** (float) A number

Return (float) The natural logarithm (base e) of the given value

Math.log10(x)

- **x** (float) A number

Return (float) The base 10 logarithm of the given value

Math.log1p(x)

- **x** (float) A number

Return (float) The natural logarithm (base e) of 1 + the given value

Math.log2(x)

- `x` (float) A number

Return (float) The base 2 logarithm of the given value

Math.max(x, y_)

- `x` (float, *optional*) A number
- `y_` (float, *optional*) Optional variable list of additional numbers

Return (float) The largest of the given numbers (zero or more)

Math.min(x, y_)

- `x` (float, *optional*) A number
- `y_` (float, *optional*) Optional variable list of additional numbers

Return (float) The smallest of the given numbers (zero or more)

Math.pow(x, y)

Calculate the power of x by y

- `x` (float) The base
- `y` (float) The exponent

Return (float) The base to the exponent power (x^y)

Math.random()

Get a random foating point number between 0 (inclusive) and 1 (exclusive)

That means a result will always be less than 1 and greater or equal to 0, the same way Math.random() works in JavaScript.

See Math.randomInt(min, max) for a function that returns random integer numbers from a given interval.

Return (float) A random floating point number between 0 (inclusive) and 1 (exclusive), that is from [0, 1)

Math.randomInt(min, max)

Get a random integer number between a min and max value (inclusive)

That means a result will always be greater than or equal to min and less than or equal to max.

- `min` (integer) The lower bound for the random number (inclusive)
- `max` (integer) The upper bound for the random number (inclusive)

Return (integer) A random number between min and max (inclusive), that is from [min, max]

Math.round(subject, precision)

Rounds the subject to the given precision

The precision defines the number of digits after the decimal point. Negative values are also supported (-1 rounds to full 10ths).

- `subject` (mixed) The value to round
- `precision` (integer, *optional*) The precision (digits after decimal point) to use, defaults to 0

Return (float) The rounded value

Math.sign(x)

Get the sign of the given number, indicating whether the number is positive, negative or zero

- `x` (integer|float) The value

Return (integer) -1, 0, 1 depending on the sign or NAN if the given value was not numeric

Math.sin(x)

- `x` (float) A number given in radians

Return (float) The sine of the given value

Math.sinh(x)

- `x` (float) A number

Return (float) The hyperbolic sine of the given value

Math.sqrt(x)

- `x` (float) A number

Return (float) The square root of the given number

Math.tan(x)

- `x` (float) A number given in radians

Return (float) The tangent of the given value

Math.tanh(x)

- `x` (float) A number

Return (float) The hyperbolic tangent of the given value

Math.trunc(x)

Get the integral part of the given number by removing any fractional digits

This function doesn't round the given number but merely calls ceil(x) or floor(x) depending on the sign of the number.

- `x` (float) A number

Return (int|float) The integral part of the given number or NAN if the given value was not numeric

1.6.11 Neos.Array

Some Functional Programming Array helpers for Eel contexts

These helpers are *WORK IN PROGRESS* and *NOT STABLE YET*

Implemented in: `Neos\Neos\Fusion\Helper\ArrayHelper`

Neos.Array.filter(set, filterProperty)

Filter an array of objects, by only keeping the elements where each object's $filterProperty evaluates to true.

- `set` (array|Collection)
- `filterProperty` (string)

Return (array)

Neos.Array.filterNegated(set, filterProperty)

Filter an array of objects, by only keeping the elements where each object's $filterProperty evaluates to false.

- `set` (array|Collection)
- `filterProperty` (string)

Return (array)

Neos.Array.groupBy(set, groupingKey)

The input is assumed to be an array or Collection of objects. Groups this input by the $groupingKey property of each element.

- `set` (array|Collection)
- `groupingKey` (string)

Return (array)

1.6.12 Neos.Backend

BackendUser helper for translations in the backend

Implemented in: `Neos\Neos\Fusion\Helper\BackendHelper`

Neos.Backend.interfaceLanguage()

The interface language the user selected or the default language defined in the settings Formatted as {@see NeosFlowI18nLocale} identifier, eg "de", "en", ...

Example:

```
Translation.id("mh").locale(Neos.Backend.interfaceLanguage()).translate()
```

Return (string)

1.6.13 Neos.Caching

Caching helper to make cache tag generation easier.

Implemented in: `Neos\Neos\Fusion\Helper\CachingHelper`

Neos.Caching.descendantOfTag(nodes)

Generate a *@cache* entry tag for descendants of a node, an array of nodes or a FlowQuery result A cache entry with this tag will be flushed whenever a node (for any variant) that is a descendant (child on any level) of one of the given nodes is updated.

- `nodes` (mixed) (A single Node or array or Traversable of Nodes)

Return (array)

Neos.Caching.nodeTag(nodes)

Generate a *@cache* entry tag for a single node, array of nodes or a FlowQuery result A cache entry with this tag will be flushed whenever one of the given nodes (for any variant) is updated.

- `nodes` (mixed) (A single Node or array or Traversable of Nodes)

Return (array)

Neos.Caching.nodeTagForIdentifier(identifier, contextNode)

Generate a *@cache* entry tag for a single node identifier. If a NodeInterface $contextNode is given the entry tag will respect the workspace hash.

- `identifier` (string)
- `contextNode` (NodeInterface|null, *optional*)

Return (string)

Neos.Caching.nodeTypeTag(nodeType, contextNode)

Generate an *@cache* entry tag for a node type A cache entry with this tag will be flushed whenever a node (for any variant) that is of the given node type(s) (including inheritance) is updated.

- `nodeType` (string|NodeType|string[]|NodeType[])
- `contextNode` (NodeInterface|null, *optional*)

Return (string|string[])

Neos.Caching.renderWorkspaceTagForContextNode(workspaceName)

- `workspaceName` (string)

Return (string)

1.6.14 Neos.Link

Eel helper for the linking service

Implemented in: `Neos\Neos\Fusion\Helper\LinkHelper`

Neos.Link.convertUriToObject(uri, contextNode)

- `uri` (string|UriInterface)
- `contextNode` (NodeInterface, *optional*)

Return (NodeInterface|AssetInterface|NULL)

Neos.Link.getScheme(uri)

- `uri` (string|UriInterface)

Return (string)

Neos.Link.hasSupportedScheme(uri)

- `uri` (string|UriInterface)

Return (boolean)

Neos.Link.resolveAssetUri(uri)

- `uri` (string|UriInterface)

Return (string)

Neos.Link.resolveNodeUri(uri, contextNode, controllerContext)

- `uri` (string|UriInterface)
- `contextNode` (NodeInterface)
- `controllerContext` (ControllerContext)

Return (string)

1.6.15 Neos.Media.Assets

This is a helper for accessing assets from the media library

Implemented in: `Neos\Media\Eel\AssetsHelper`

Neos.Media.Assets.findByCollection(collection)

Return (QueryResultInterface<AssetInterface>) | null

Neos.Media.Assets.findByTag(tag)

Return (QueryResultInterface) | null

Neos.Media.Assets.search(searchTerm, tags, collection)

- `tags` (Tag[]|string[], *optional*)

Return (QueryResultInterface<AssetInterface>) | null

1.6.16 Neos.Node

Eel helper for ContentRepository Nodes

Implemented in: `Neos\Neos\Fusion\Helper\NodeHelper`

Neos.Node.isOfType(node, nodeType)

If this node type or any of the direct or indirect super types has the given name.

- `node` (NodeInterface)
- `nodeType` (string)

Return (bool)

Neos.Node.labelForNode(node)

Generate a label for a node with a chaining mechanism. To be used in nodetype definitions.

- `node` (NodeInterface|null, *optional*)

Return (NodeLabelToken)

Neos.Node.nearestContentCollection(node, nodePath)

Check if the given node is already a collection, find collection by nodePath otherwise, throw exception if no content collection could be found

- `node` (NodeInterface)
- `nodePath` (string)

Return (NodeInterface)

1.6.17 Neos.Rendering

Render Content Dimension Names, Node Labels

These helpers are *WORK IN PROGRESS* and *NOT STABLE YET*

Implemented in: `Neos\Neos\Fusion\Helper\RenderingHelper`

Neos.Rendering.injectConfigurationManager(configurationManager)

- `configurationManager` (ConfigurationManager)

Return (void)

Neos.Rendering.labelForNodeType(nodeTypeName)

Render the label for the given $nodeTypeName

- `nodeTypeName` (string)

Return (string)

Neos.Rendering.renderDimensions(dimensions)

Render a human-readable description for the passed $dimensions

- `dimensions` (array)

Return (string)

1.6.18 Neos.Seo.Image

Implemented in: `Neos\Seo\Fusion\Helper\ImageHelper`

Neos.Seo.Image.createThumbnail(asset, preset, width, maximumWidth, height, maximumHeight, allowCropping, allowUpScaling, async, quality, format)

- `asset` (AssetInterface)
- `preset` (string, *optional*) Name of the preset that should be used as basis for the configuration
- `width` (integer, *optional*) Desired width of the image
- `maximumWidth` (integer, *optional*) Desired maximum width of the image
- `height` (integer, *optional*) Desired height of the image
- `maximumHeight` (integer, *optional*) Desired maximum height of the image
- `allowCropping` (boolean, *optional*) Whether the image should be cropped if the given sizes would hurt the aspect ratio
- `allowUpScaling` (boolean, *optional*) Whether the resulting image size might exceed the size of the original image
- `async` (boolean, *optional*) Whether the thumbnail can be generated asynchronously
- `quality` (integer, *optional*) Quality of the processed image
- `format` (string, *optional*) Format for the image, only jpg, jpeg, gif, png, wbmp, xbm, webp and bmp are supported.

Return (null|ImageInterface)

1.6.19 Neos.Ui.PositionalArraySorter

Implemented in: `Neos\Neos\Ui\Fusion\Helper\PositionalArraySorterHelper`

Neos.Ui.PositionalArraySorter.sort(array, positionPath)

- `array` (array)
- `positionPath` (string, *optional*)

Return (array)

1.6.20 Neos.Ui.StaticResources

Implemented in: `Neos\Neos\Ui\Fusion\Helper\StaticResourcesHelper`

Neos.Ui.StaticResources.compiledResourcePackage()

1.6.21 Neos.Ui.Workspace

Implemented in: `Neos\Neos\Ui\Fusion\Helper\WorkspaceHelper`

Neos.Ui.Workspace.getAllowedTargetWorkspaces()

Neos.Ui.Workspace.getPersonalWorkspace()

Neos.Ui.Workspace.getPublishableNodeInfo(workspace)

- `workspace` (Workspace)

Return (array)

1.6.22 NodeInfo

Implemented in: `Neos\Neos\Ui\Fusion\Helper\NodeInfoHelper`

NodeInfo.createRedirectToNode(controllerContext, node)

Creates a URL that will redirect to the given $node in live or base workspace, or returns an empty string if that doesn't exist or is inaccessible

- `controllerContext` (ControllerContext)
- `node` (NodeInterface|null, *optional*)

Return (string)

NodeInfo.defaultNodesForBackend(site, documentNode, controllerContext)

- `site` (NodeInterface)
- `documentNode` (NodeInterface)
- `controllerContext` (ControllerContext)

Return (array)

NodeInfo.renderDocumentNodeAndChildContent(documentNode, controllerContext)

- `documentNode` (NodeInterface)
- `controllerContext` (ControllerContext)

Return (array)

NodeInfo.renderNodeWithMinimalPropertiesAndChildrenInformation(node, controllerContext, nodeTypeFilterOverride)

- `node` (NodeInterface)
- `controllerContext` (ControllerContext|null, *optional*)
- `nodeTypeFilterOverride` (string, *optional*)

Return (array|null)

NodeInfo.renderNodeWithPropertiesAndChildrenInformation(node, controllerContext, nodeTypeFilterOverride)

- `node` (NodeInterface)
- `controllerContext` (ControllerContext|null, *optional*)
- `nodeTypeFilterOverride` (string, *optional*)

Return (array|null)

NodeInfo.renderNodes(nodes, controllerContext, omitMostPropertiesForTreeState)

- `nodes` (array)
- `controllerContext` (ControllerContext)
- `omitMostPropertiesForTreeState` (bool, *optional*)

Return (array)

NodeInfo.renderNodesWithParents(nodes, controllerContext)

- `nodes` (array)
- `controllerContext` (ControllerContext)

Return (array)

NodeInfo.uri(node, controllerContext)

- `node` (NodeInterface)
- `controllerContext` (ControllerContext)

Return (string)

1.6.23 Security

Helper for security related information

Implemented in: `Neos\Eel\Helper\SecurityHelper`

Security.csrfToken()

Returns CSRF token which is required for "unsafe" requests (e.g. POST, PUT, DELETE, ...)

Return (string)

Security.getAccount()

Get the account of the first authenticated token.

Return (Account|NULL)

Security.hasAccess(privilegeTarget, parameters)

Returns true, if access to the given privilege-target is granted

- `privilegeTarget` (string) The identifier of the privilege target to decide on
- `parameters` (array, *optional*) Optional array of privilege parameters (simple key => value array)

Return (boolean) true if access is granted, false otherwise

Security.hasRole(roleIdentifier)

Returns true, if at least one of the currently authenticated accounts holds a role with the given identifier, also recursively.

- `roleIdentifier` (string) The string representation of the role to search for

Return (boolean) true, if a role with the given string representation was found

Security.isAuthenticated()

Returns true, if any account is currently authenticated

Return (boolean) true if any account is authenticated

1.6.24 StaticResource

Implemented in: `Neos\Flow\ResourceManagement\EelHelper\StaticResourceHelper`

StaticResource.content(packageKey, pathAndFilename, localize)

Get the content of a package resource

- `packageKey` (string) Package key where the resource is from.
- `pathAndFilename` (string) The path and filename of the resource. Starting with "Public/…" or "Private/…
- `localize` (bool, *optional*) If enabled localizing of the resource is attempted by adding locales from the current locale-chain between filename and extension.

Return (string)

StaticResource.uri(packageKey, pathAndFilename, localize)

Get the public uri of a package resource

- `packageKey` (string) Package key where the resource is from.
- `pathAndFilename` (string) The path and filename of the resource. Has to start with "Public/…" as private resources do not have a uri.
- `localize` (bool, *optional*) If enabled localizing of the resource is attempted by adding locales from the current locale-chain between filename and extension.

Return (string)

1.6.25 String

String helpers for Eel contexts

Implemented in: `Neos\Eel\Helper\StringHelper`

String.base64decode(string, strict)

Implementation of the PHP base64_decode function

- `string` (string) The encoded data.
- `strict` (bool, *optional*) If TRUE this function will return FALSE if the input contains character from outside the base64 alphabet.

Return (string|bool) The decoded data or FALSE on failure. The returned data may be binary.

String.base64encode(string)

Implementation of the PHP base64_encode function

- `string` (string) The data to encode.

Return (string) The encoded data

String.charAt(string, index)

Get the character at a specific position

Example:

```
String.charAt("abcdefg", 5) == "f"
```

- `string` (string) The input string
- `index` (integer) The index to get

Return (string) The character at the given index

String.chr(value)

Generate a single-byte string from a number

Example:

```
String.chr(65) == "A"
```

This is a wrapper for the chr() PHP function.

- `value` (int) An integer between 0 and 255

Return (string) A single-character string containing the specified byte

String.crop(string, maximumCharacters, suffix)

Crop a string to `maximumCharacters` length, optionally appending `suffix` if cropping was necessary.

- `string` (string) The input string
- `maximumCharacters` (integer) Number of characters where cropping should happen
- `suffix` (string, *optional*) Suffix to be appended if cropping was necessary

Return (string) The cropped string

String.cropAtSentence(string, maximumCharacters, suffix)

Crop a string to `maximumCharacters` length, taking sentences into account, optionally appending `suffix` if cropping was necessary.

- `string` (string) The input string
- `maximumCharacters` (integer) Number of characters where cropping should happen
- `suffix` (string, *optional*) Suffix to be appended if cropping was necessary

Return (string) The cropped string

String.cropAtWord(string, maximumCharacters, suffix)

Crop a string to `maximumCharacters` length, taking words into account, optionally appending `suffix` if cropping was necessary.

- `string` (string) The input string
- `maximumCharacters` (integer) Number of characters where cropping should happen
- `suffix` (string, *optional*) Suffix to be appended if cropping was necessary

Return (string) The cropped string

String.endsWith(string, search, position)

Test if a string ends with the given search string

Example:

```
String.endsWith('Hello, World!', 'World!') == true
```

- `string` (string) The string
- `search` (string) A string to search
- `position` (int|null, *optional*) Optional position for limiting the string

Return (boolean) true if the string ends with the given search

String.firstLetterToLowerCase(string)

Lowercase the first letter of a string

Example:

```
String.firstLetterToLowerCase('CamelCase') == 'camelCase'
```

- `string` (string) The input string

Return (string) The string with the first letter in lowercase

String.firstLetterToUpperCase(string)

Uppercase the first letter of a string

Example:

```
String.firstLetterToUpperCase('hello world') == 'Hello world'
```

- `string` (string) The input string

Return (string) The string with the first letter in uppercase

String.format(format, args)

Implementation of the PHP vsprintf function

- `format` (string) A formatting string containing directives
- `args` (array) An array of values to be inserted according to the formatting string $format

Return (string) A string produced according to the formatting string $format

String.htmlSpecialChars(string, preserveEntities)

Convert special characters to HTML entities

- `string` (string) The string to convert
- `preserveEntities` (boolean, *optional*) `true` if entities should not be double encoded

Return (string) The converted string

String.indexOf(string, search, fromIndex)

Find the first position of a substring in the given string

Example:

```
String.indexOf("Blue Whale", "Blue") == 0
```

- `string` (string) The input string
- `search` (string) The substring to search for
- `fromIndex` (integer, *optional*) The index where the search should start, defaults to the beginning

Return (integer) The index of the substring (>= 0) or -1 if the substring was not found

String.isBlank(string)

Test if the given string is blank (empty or consists of whitespace only)

Examples:

```
String.isBlank('') == true
String.isBlank('  ') == true
```

- `string` (string) The string to test

Return (boolean) `true` if the given string is blank

String.lastIndexOf(string, search, toIndex)

Find the last position of a substring in the given string

Example:

```
String.lastIndexOf("Developers Developers Developers!", "Developers") == 22
```

- `string` (string) The input string
- `search` (string) The substring to search for
- `toIndex` (integer, *optional*) The position where the backwards search should start, defaults to the end

Return (integer) The last index of the substring (>=0) or -1 if the substring was not found

String.length(string)

Get the length of a string

- `string` (string) The input string

Return (integer) Length of the string

String.md5(string)

Calculate the MD5 checksum of the given string

Example:

```
String.md5("joh316") == "bacb98acf97e0b6112b1d1b650b84971"
```

- `string` (string) The string to hash

Return (string) The MD5 hash of `string`

String.nl2br(string)

Insert HTML line breaks before all newlines in a string

Example:

```
String.nl2br(someStingWithLinebreaks) == 'line1<br />line2'
```

This is a wrapper for the nl2br() PHP function.

- `string` (string) The input string

Return (string) The string with new lines replaced

String.ord(string)

Convert the first byte of a string to a value between 0 and 255

Example:

```
String.ord('A') == 65
```

This is a wrapper for the ord() PHP function.

- `string` (string) A character

Return (int) An integer between 0 and 255

String.pregMatch(string, pattern)

Match a string with a regular expression (PREG style)

Example:

```
String.pregMatch("For more information, see Chapter 3.4.5.1", "/(chapter \d+(\.\d)*)/i")
  == ['Chapter 3.4.5.1', 'Chapter 3.4.5.1', '.1']
```

- `string` (string) The input string
- `pattern` (string) A PREG pattern

Return (array) The matches as array or NULL if not matched

String.pregMatchAll(string, pattern)

Perform a global regular expression match (PREG style)

Example:

```
String.pregMatchAll("<hr id="icon-one" /><hr id="icon-two" />", '/id="icon-(.+?)"/')
  == [['id="icon-one"', 'id="icon-two"'],['one','two']]
```

- `string` (string) The input string
- `pattern` (string) A PREG pattern

Return (array) The matches as array or NULL if not matched

String.pregReplace(string, pattern, replace, limit)

Replace occurrences of a search string inside the string using regular expression matching (PREG style)

Examples:

```
String.pregReplace("Some.String with sp:cial characters", "/[[:^alnum:]]/", "-") ==
↪"Some-String-with-sp-cial-characters"
String.pregReplace("Some.String with sp:cial characters", "/[[:^alnum:]]/", "-", 1) ==
↪"Some-String with sp:cial characters"
String.pregReplace("2016-08-31", "/([0-9]+)-([0-9]+)-([0-9]+)/", "$3.$2.$1") == "31.08.
↪2016"
```

- `string` (string) The input string

- `pattern` (string) A PREG pattern
- `replace` (string) A replacement string, can contain references to capture groups with "\n" or "$n
- `limit` (integer, *optional*) The maximum possible replacements for each pattern in each subject string. Defaults to -1 (no limit).

Return (string) The string with all occurrences replaced

String.pregSplit(string, pattern, limit)

Split a string by a separator using regular expression matching (PREG style)

Examples:

```
String.pregSplit("foo bar   baz", "/\s+/") == ['foo', 'bar', 'baz']
String.pregSplit("first second third", "/\s+/", 2) == ['first', 'second third']
```

- `string` (string) The input string
- `pattern` (string) A PREG pattern
- `limit` (integer, *optional*) The maximum amount of items to return, in contrast to split() this will return all remaining characters in the last item (see example)

Return (array) An array of the splitted parts, excluding the matched pattern

String.rawUrlDecode(string)

Decode the string from URLs according to RFC 3986

- `string` (string) The string to decode

Return (string) The decoded string

String.rawUrlEncode(string)

Encode the string for URLs according to RFC 3986

- `string` (string) The string to encode

Return (string) The encoded string

String.replace(string, search, replace)

Replace occurrences of a search string inside the string

Example:

```
String.replace("canal", "ana", "oo") == "cool"
String.replace("cool gridge", ["oo", "gri"], ["ana", "bri"]) == "canal bridge"
```

Note: this method does not perform regular expression matching, @see pregReplace().

- `string` (array|string|null) The input string
- `search` (array|string|null) A search string
- `replace` (array|string|null) A replacement string

Return (array|string|string[]) The string with all occurrences replaced

String.sha1(string)

Calculate the SHA1 checksum of the given string

Example:

```
String.sha1("joh316") == "063b3d108bed9f88fa618c6046de0dccadcf3158"
```

- `string` (string) The string to hash

Return (string) The SHA1 hash of `string`

String.split(string, separator, limit)

Split a string by a separator

Example:

```
String.split("My hovercraft is full of eels", " ") == ['My', 'hovercraft', 'is', 'full',
↪'of', 'eels']
String.split("Foo", "", 2) == ['F', 'o']
```

Node: This implementation follows JavaScript semantics without support of regular expressions.

- `string` (string) The string to split
- `separator` (string|null, *optional*) The separator where the string should be splitted
- `limit` (int|null, *optional*) The maximum amount of items to split (exceeding items will be discarded)

Return (array) An array of the splitted parts, excluding the separators

String.startsWith(string, search, position)

Test if a string starts with the given search string

Examples:

```
String.startsWith('Hello world!', 'Hello') == true
String.startsWith('My hovercraft is full of...', 'Hello') == false
String.startsWith('My hovercraft is full of...', 'hovercraft', 3) == true
```

- `string` (string) The input string
- `search` (string) The string to search for
- `position` (integer, *optional*) The position to test (defaults to the beginning of the string)

Return (boolean)

String.stripTags(string, allowableTags)

Strip all HTML tags from the given string

Example:

```
String.stripTags('<a href="#">Some link</a>') == 'Some link'
```

This is a wrapper for the strip_tags() PHP function.

- `string` (string) The string to strip
- `allowableTags` (string|null, *optional*) Specify tags which should not be stripped

Return (string) The string with tags stripped

String.substr(string, start, length)

Return the characters in a string from start up to the given length

This implementation follows the JavaScript specification for "substr".

Examples:

```
String.substr('Hello, World!', 7, 5) == 'World'
String.substr('Hello, World!', 7) == 'World!'
String.substr('Hello, World!', -6) == 'World!'
```

- `string` (string) A string
- `start` (integer) Start offset
- `length` (integer, *optional*) Maximum length of the substring that is returned

Return (string) The substring

String.substring(string, start, end)

Return the characters in a string from a start index to an end index

This implementation follows the JavaScript specification for "substring".

Examples:

```
String.substring('Hello, World!', 7, 12) == 'World'
String.substring('Hello, World!', 7) == 'World!'
```

- `string` (string)
- `start` (integer) Start index
- end (integer, *optional*) End index

Return (string) The substring

String.toBoolean(string)

Convert a string to boolean

A value is `true`, if it is either the string `"true"` or `"TRUE"` or the number 1.

- `string` (string) The string to convert

Return (boolean) The boolean value of the string (`true` or `false`)

String.toFloat(string)

Convert a string to float

- `string` (string) The string to convert

Return (float) The float value of the string

String.toInteger(string)

Convert a string to integer

- `string` (string) The string to convert

Return (integer) The converted string

String.toLowerCase(string)

Lowercase a string

- `string` (string) The input string

Return (string) The string in lowercase

String.toString(value)

Convert the given value to a string

- `value` (mixed) The value to convert (must be convertible to string)

Return (string) The string value

String.toUpperCase(string)

Uppercase a string

- `string` (string) The input string

Return (string) The string in uppercase

String.trim(string, charlist)

Trim whitespace at the beginning and end of a string

- `string` (string) The string to trim
- `charlist` (string, *optional*) List of characters that should be trimmed, defaults to whitespace

Return (string) The trimmed string

String.wordCount(unicodeString)

Return the count of words for a given string. Remove marks & digits and flatten all kind of whitespaces (tabs, new lines and multiple spaces) For example this helper can be utilized to calculate the reading time of an article.

- `unicodeString` (string) The input string

Return (integer) Number of words

1.6.26 Translation

Translation helpers for Eel contexts

Implemented in: `Neos\Flow\I18n\EelHelper\TranslationHelper`

Translation.id(id)

Start collection of parameters for translation by id

- `id` (string) Id to use for finding translation (trans-unit id in XLIFF)

Return (TranslationParameterToken)

Translation.translate(id, originalLabel, arguments, source, package, quantity, locale)

Get the translated value for an id or original label

If only id is set and contains a translation shorthand string, translate according to that shorthand

In all other cases:

Replace all placeholders with corresponding values if they exist in the translated label.

- `id` (string) Id to use for finding translation (trans-unit id in XLIFF)
- `originalLabel` (string, *optional*) The original translation value (the untranslated source string).
- `arguments` (array, *optional*) Array of numerically indexed or named values to be inserted into placeholders. Have a look at the internationalization documentation in the definitive guide for details.
- `source` (string, *optional*) Name of file with translations
- `package` (string, *optional*) Target package key. If not set, the current package key will be used
- `quantity` (mixed, *optional*) A number to find plural form for (float or int), NULL to not use plural forms
- `locale` (string, *optional*) An identifier of locale to use (NULL for use the default locale)

Return (string|null) Translated label or source label / ID key

Translation.value(value)

Start collection of parameters for translation by original label

- `value` (string)

Return (TranslationParameterToken)

1.6.27 Type

Type helper for Eel contexts

Implemented in: `Neos\Eel\Helper\TypeHelper`

Type.className(variable)

Get the class name of the given variable or NULL if it wasn't an object

- `variable` (object)

Return (string|NULL)

Type.debugType(variable)

Get the classname for objects or type for other values

- `variable` (mixed)

Return (string)

Type.getType(variable)

Get the variable type

- `variable` (mixed)

Return (string)

Type.instance(variable, expectedObjectType)

Is the given variable of the provided object type.

- `variable` (mixed)
- `expectedObjectType` (string)

Return (boolean)

Type.isArray(variable)

Is the given variable an array.

- `variable` (mixed)

Return (boolean)

Type.isBoolean(variable)

Is the given variable boolean.

- `variable` (mixed)

Return (boolean)

Type.isFloat(variable)

Is the given variable a float.

- `variable` (mixed)

Return (boolean)

Type.isInteger(variable)

Is the given variable an integer.

- `variable` (mixed)

Return (boolean)

Type.isNumeric(variable)

Is the given variable numeric.

- `variable` (mixed)

Return (boolean)

Type.isObject(variable)

Is the given variable an object.

- `variable` (mixed)

Return (boolean)

Type.isScalar(variable)

Is the given variable a scalar.

- `variable` (mixed)

Return (boolean)

Type.isString(variable)

Is the given variable a string.

- `variable` (mixed)

Return (boolean)

Type.typeof(variable)

Get the variable type

- `variable` (mixed)

Return (string)

1.7 FlowQuery Operation Reference

This reference was automatically generated from code on 2024-03-10

1.7.1 add

Adds the given items to the current context. The operation accepts one argument that may be an Array, a FlowQuery or an Object.

Implementation
Neos\Eel\FlowQuery\Operations\AddOperation

Priority
1

Final
No

Returns
void

1.7.2 cacheLifetime

"cacheLifetime" operation working on ContentRepository nodes. Will get the minimum of all allowed cache lifetimes for the nodes in the current FlowQuery context. This means it will evaluate to the nearest future value of the hiddenBeforeDateTime or hiddenAfterDateTime properties of all nodes in the context. If none are set or all values are in the past it will evaluate to NULL.

To include already hidden nodes (with a hiddenBeforeDateTime value in the future) in the result, also invisible nodes have to be included in the context. This can be achieved using the "context" operation before fetching child nodes.

Example:

q(node).context({'invisibleContentShown': true}).children().cacheLifetime()

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\CacheLifetimeOperation

Priority
1

Final
Yes

Returns
integer The cache lifetime in seconds or NULL if either no content collection was given or no child node had a "hiddenBeforeDateTime" or "hiddenAfterDateTime" property set

1.7.3 children

"children" operation working on ContentRepository nodes. It iterates over all context elements and returns all child nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ChildrenOperation

Priority
100

Final
No

Returns
void

1.7.4 children

"children" operation working on generic objects. It iterates over all context elements and returns the values of the properties given in the filter expression that has to be specified as argument or in a following filter operation.

Implementation
Neos\Eel\FlowQuery\Operations\Object\ChildrenOperation

Priority
1

Final
No

Returns
void

1.7.5 closest

"closest" operation working on ContentRepository nodes. For each node in the context, get the first node that matches the selector by testing the node itself and traversing up through its ancestors.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ClosestOperation

Priority
100

Final
No

Returns
void

1.7.6 context

"context" operation working on ContentRepository nodes. Modifies the ContentRepository Context of each node in the current FlowQuery context by the given properties and returns the same nodes by identifier if they can be accessed in the new Context (otherwise they will be skipped).

Example:

q(node).context({'invisibleContentShown': true}).children()

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ContextOperation

Priority
1

Final
No

Returns
void

1.7.7 count

Count the number of elements in the context.

If arguments are given, these are used to filter the elements before counting.

Implementation
Neos\Eel\FlowQuery\Operations\CountOperation

Priority
1

Final
Yes

Returns
void|integer with the number of elements

1.7.8 filter

Filter operation, limiting the set of objects. The filter expression is expected as string argument and used to reduce the context to matching elements by checking each value against the filter.

A filter expression is written in Fizzle, a grammar inspired by CSS selectors. It has the form *"[" [<value>] <operator> <operand> "]"* and supports the following operators:

=~
: Strict equality of case-insensitive value and operand

=
: Strict equality of value and operand

!=~
: Strict inequality of case-insensitive value and operand

!=
: Strict inequality of value and operand

<
: Value is less than operand

<=
: Value is less than or equal to operand

>
: Value is greater than operand

>=
: Value is greater than or equal to operand

$=~
: Value ends with operand (string-based) or case-insensitive value's last element is equal to operand (array-based)

$=
: Value ends with operand (string-based) or value's last element is equal to operand (array-based)

^=~
: Value starts with operand (string-based) or case-insensitive value's first element is equal to operand (array-based)

^=
: Value starts with operand (string-based) or value's first element is equal to operand (array-based)

***=~**
: Value contains operand (string-based) or case-insensitive value contains an element that is equal to operand (array based)

***=**
: Value contains operand (string-based) or value contains an element that is equal to operand (array based)

instanceof
: Checks if the value is an instance of the operand

!instanceof
: Checks if the value is not an instance of the operand

For the latter the behavior is as follows: if the operand is one of the strings object, array, int(eger), float, double, bool(ean) or string the value is checked for being of the specified type. For any other strings the value is used as classname with the PHP instanceof operation to check if the value matches.

Implementation
: Neos\Eel\FlowQuery\Operations\Object\FilterOperation

Priority
1

Final
No

Returns
void

1.7.9 filter

This filter implementation contains specific behavior for use on ContentRepository nodes. It will not evaluate any elements that are not instances of the *NodeInterface*.

The implementation changes the behavior of the *instanceof* operator to work on node types instead of PHP object types, so that:

```
[instanceof Acme.Com:Page]
```

will in fact use *isOfType()* on the *NodeType* of context elements to filter. This filter allow also to filter the current context by a given node. Anything else remains unchanged.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\FilterOperation

Priority
100

Final
No

Returns
void

1.7.10 find

"find" operation working on ContentRepository nodes. This operation allows for retrieval of nodes specified by a path, identifier or node type (recursive).

Example (node name):

q(node).find('main')

Example (relative path):

q(node).find('main/text1')

Example (absolute path):

q(node).find('/sites/my-site/home')

Example (identifier):

q(node).find('#30e893c1-caef-0ca5-b53d-e5699bb8e506')

Example (node type):

q(node).find('[instanceof Acme.Com:Text]')

Example (multiple node types):

q(node).find('[instanceof Acme.Com:Text],[instanceof Acme.Com:Image]')

Example (node type with filter):

q(node).find('[instanceof Acme.Com:Text][text*="Neos"]')

This operation operates rather on the given Context object than on the given node and thus may work with the legacy node interface until subgraphs are available {@inheritdoc}

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\FindOperation

Priority
100

Final
No

Returns
void

1.7.11 first

Get the first element inside the context.

Implementation
Neos\Eel\FlowQuery\Operations\FirstOperation

Priority
1

Final
No

Returns
void

1.7.12 get

Get a (non-wrapped) element from the context.

If FlowQuery is used, the result is always another FlowQuery. In case you need to pass a FlowQuery result (and lazy evaluation does not work out) you can use get() to unwrap the result from the "FlowQuery envelope".

If no arguments are given, the full context is returned. Otherwise the value contained in the context at the index given as argument is returned. If no such index exists, NULL is returned.

Implementation
Neos\Eel\FlowQuery\Operations\GetOperation

Priority
1

Final
Yes

Returns
mixed

1.7.13 has

"has" operation working on NodeInterface. Reduce the set of matched elements to those that have a child node that matches the selector or given subject.

Accepts a selector, an array, an object, a traversable object & a FlowQuery object as argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\HasOperation

Priority
100

Final
No

Returns
void

1.7.14 is

Check whether the at least one of the context elements match the given filter.

Without arguments is evaluates to true if the context is not empty. If arguments are given, they are used to filter the context before evaluation.

Implementation
Neos\Eel\FlowQuery\Operations\IsOperation

Priority
1

Final
Yes

Returns
void|boolean

1.7.15 last

Get the last element inside the context.

Implementation
Neos\Eel\FlowQuery\Operations\LastOperation

Priority
1

Final
No

Returns
void

1.7.16 neosUiDefaultNodes

Fetches all nodes needed for the given state of the UI

Implementation
Neos\Neos\Ui\FlowQueryOperations\NeosUiDefaultNodesOperation

Priority
100

Final
No

Returns
void

1.7.17 neosUiFilteredChildren

"children" operation working on ContentRepository nodes. It iterates over all context elements and returns all child nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\Neos\Ui\FlowQueryOperations\NeosUiFilteredChildrenOperation

Priority
100

Final
No

Returns
void

1.7.18 next

"next" operation working on ContentRepository nodes. It iterates over all context elements and returns the immediately following sibling. If an optional filter expression is provided, it only returns the node if it matches the given expression.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\NextOperation

Priority
100

Final
No

Returns
void

1.7.19 nextAll

"nextAll" operation working on ContentRepository nodes. It iterates over all context elements and returns each following sibling or only those matching the filter expression specified as optional argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\NextAllOperation

Priority
0

Final
No

Returns
void

1.7.20 nextUntil

"nextUntil" operation working on ContentRepository nodes. It iterates over all context elements and returns each following sibling until the matching sibling is found. If an optional filter expression is provided as a second argument, it only returns the nodes matching the given expression.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\NextUntilOperation

Priority
0

Final
No

Returns
void

1.7.21 parent

"parent" operation working on ContentRepository nodes. It iterates over all context elements and returns each direct parent nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ParentOperation

Priority
100

Final
No

Returns
void

1.7.22 parents

"parents" operation working on ContentRepository nodes. It iterates over all context elements and returns the parent nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ParentsOperation

Priority
0

Final
No

Returns
void

1.7.23 parents

"parents" operation working on ContentRepository nodes. It iterates over all context elements and returns the parent nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\Neos\Eel\FlowQueryOperations\ParentsOperation

Priority
100

Final
No

Returns
void

1.7.24 parentsUntil

"parentsUntil" operation working on ContentRepository nodes. It iterates over all context elements and returns the parent nodes until the matching parent is found. If an optional filter expression is provided as a second argument, it only returns the nodes matching the given expression.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\ParentsUntilOperation

Priority
0

Final
No

Returns
void

1.7.25 parentsUntil

"parentsUntil" operation working on ContentRepository nodes. It iterates over all context elements and returns the parent nodes until the matching parent is found. If an optional filter expression is provided as a second argument, it only returns the nodes matching the given expression.

Implementation
Neos\Neos\Eel\FlowQueryOperations\ParentsUntilOperation

Priority
100

Final
No

Returns
void

1.7.26 prev

"prev" operation working on ContentRepository nodes. It iterates over all context elements and returns the immediately preceding sibling. If an optional filter expression is provided, it only returns the node if it matches the given expression.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\PrevOperation

Priority
100

Final
No

Returns
void

1.7.27 prevAll

"prevAll" operation working on ContentRepository nodes. It iterates over all context elements and returns each preceding sibling or only those matching the filter expression specified as optional argument

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\PrevAllOperation

Priority
0

Final
No

Returns
void

1.7.28 prevUntil

"prevUntil" operation working on ContentRepository nodes. It iterates over all context elements and returns each preceding sibling until the matching sibling is found. If an optional filter expression is provided as a second argument, it only returns the nodes matching the given expression.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\PrevUntilOperation

Priority
0

Final
No

Returns
void

1.7.29 property

Used to access properties of a ContentRepository Node. If the property mame is prefixed with _, internal node properties like start time, end time, hidden are accessed.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\PropertyOperation

Priority
100

Final
Yes

Returns
mixed

1.7.30 property

Access properties of an object using ObjectAccess.

Expects the name of a property as argument. If the context is empty, NULL is returned. Otherwise the value of the property on the first context element is returned.

Implementation
Neos\Eel\FlowQuery\Operations\Object\PropertyOperation

Priority
1

Final
Yes

Returns
mixed

1.7.31 remove

Removes the given items from the current context. The operation accepts one argument that may be an Array, a FlowQuery or an Object.

Implementation
Neos\Eel\FlowQuery\Operations\RemoveOperation

Priority
1

Final
No

Returns
void

1.7.32 search

Implementation
Neos\Neos\Ui\FlowQueryOperations\SearchOperation

Priority
100

Final
No

Returns
void

1.7.33 siblings

"siblings" operation working on ContentRepository nodes. It iterates over all context elements and returns all sibling nodes or only those matching the filter expression specified as optional argument.

Implementation
Neos\ContentRepository\Eel\FlowQueryOperations\SiblingsOperation

Priority
100

Final
No

Returns
void

1.7.34 slice

Slice the current context

If no arguments are given, the full context is returned. Otherwise the value contained in the context are sliced with offset and length.

Implementation
Neos\Eel\FlowQuery\Operations\SliceOperation

Priority
1

Final
No

Returns
void

1.7.35 sort

"sort" operation working on ContentRepository nodes. Sorts nodes by specified node properties.

{@inheritdoc}

First argument is the node property to sort by. Works with internal arguments (_xyz) as well. Second argument is the sort direction (ASC or DESC). Third optional argument are the sort options (see https://www.php.net/manual/en/function.sort):

- 'SORT_REGULAR'
- 'SORT_NUMERIC'
- 'SORT_STRING'
- 'SORT_LOCALE_STRING'
- 'SORT_NATURAL'
- 'SORT_FLAG_CASE' (use as last option with SORT_STRING, SORT_LOCALE_STRING or SORT_NATURAL)

A single sort option can be supplied as string. Multiple sort options are supplied as array. Other than the above listed sort options throw an error. Omitting the third parameter leaves FlowQuery sort() in SORT_REGULAR sort mode. Example usages:

sort("title", "ASC", ["SORT_NATURAL", "SORT_FLAG_CASE"]) sort("risk", "DESC", "SORT_NUMERIC")

Implementation
Neos\Neos\Eel\FlowQueryOperations\SortOperation

Priority
1

Final
No

Returns
void

1.8 Neos Command Reference

The commands in this reference are shown with their full command identifiers. On your system you can use shorter identifiers, whose availability depends on the commands available in total (to avoid overlap the shortest possible identifier is determined during runtime).

To see the shortest possible identifiers on your system as well as further commands that may be available, use:

```
./flow help
```

The following reference was automatically generated from code on 2024-03-10

1.8.1 Package *NEOS.CONTENTREPOSITORY*

neos.contentrepository:node:repair

Repair inconsistent nodes

This command analyzes and repairs the node tree structure and individual nodes based on the current node type configuration.

It is possible to execute only one or more specific checks by providing the **–skip** or **–only** option. See the full description of checks further below for possible check identifiers.

The following checks will be performed:

Remove abstract and undefined node types removeAbstractAndUndefinedNodes

Will remove all nodes that has an abstract or undefined node type.

Remove orphan (parentless) nodes removeOrphanNodes

Will remove all child nodes that do not have a connection to the root node.

Remove disallowed child nodes removeDisallowedChildNodes

Will remove all child nodes that are disallowed according to the node type's auto-create configuration and constraints.

Remove undefined node properties removeUndefinedProperties

Remove broken object references removeBrokenEntityReferences

Detects and removes references from nodes to entities which don't exist anymore (for example Image nodes referencing ImageVariant objects which are gone for some reason).

Will remove all undefined properties according to the node type configuration.

Remove nodes with invalid dimensions removeNodesWithInvalidDimensions

Will check for and optionally remove nodes which have dimension values not matching the current content dimension configuration.

Remove nodes with invalid workspace removeNodesWithInvalidWorkspace

Will check for and optionally remove nodes which belong to a workspace which no longer exists..

Repair inconsistent node identifiers fixNodesWithInconsistentIdentifier

Will check for and optionally repair node identifiers which are out of sync with their corresponding nodes in a live workspace.

Missing child nodes createMissingChildNodes

For all nodes (or only those which match the –node-type filter specified with this command) which currently don't have child nodes as configured by the node type's configuration new child nodes will be created.

Reorder child nodes reorderChildNodes

For all nodes (or only those which match the –node-type filter specified with this command) which have configured child nodes, those child nodes are reordered according to the position from the parents NodeType configuration.

Missing default properties addMissingDefaultValues

For all nodes (or only those which match the –node-type filter specified with this command) which currently dont have a property that have a default value configuration the default value for that property will be set.

Repair nodes with missing shadow nodes repairShadowNodes

This will reconstruct missing shadow nodes in case something went wrong in creating or publishing them. This must be used on a workspace other than live.

It searches for nodes which have a corresponding node in one of the base workspaces, have different node paths, but don't have a corresponding shadow node with a "movedto" value.

Create missing sites node createMissingSitesNode

If needed, creates a missing "/sites" node, which is essential for Neos to work properly.

Generate missing URI path segments generateUriPathSegments

Generates URI path segment properties for all document nodes which don't have a path segment set yet.

Remove content dimensions from / and /sites removeContentDimensionsFromRootAndSitesNode

Removes content dimensions from the root and sites nodes

Examples:

./flow node:repair

./flow node:repair –node-type Acme.Com:Page

./flow node:repair –workspace user-robert –only removeOrphanNodes,removeNodesWithInvalidDimensions

./flow node:repair –skip removeUndefinedProperties

Options

`--node-type`
: Node type name, if empty update all declared node types

`--workspace`
: Workspace name, default is 'live'

`--dry-run`
: Don't do anything, but report actions

`--cleanup`
: If false, cleanup tasks are skipped

`--skip`
: Skip the given check or checks (comma separated)

`--only`
: Only execute the given check or checks (comma separated)

`neos.contentrepository:nodetypes:list`

Lists all declared NodeTypes grouped by namespace

Options

`--filter`
: Only NodeType-names containing this string will be listed

`--include-abstract`
: List abstract NodeTypes

`neos.contentrepository:nodetypes:show`

Shows the merged configuration (including supertypes) of a NodeType

Arguments

`--node-type-name`
: The name of the NodeType to show

Options

`--path`
: Path of the NodeType-configuration which will be shown

`--level`
: Truncate the configuration at this depth and show '...' (Usefully for only seeing the keys of the properties)

1.8.2 Package *NEOS.FLOW*

`neos.flow:cache:collectgarbage`

Cache Garbage Collection

Runs the Garbage Collection (collectGarbage) method on all registered caches.

Though the method is defined in the BackendInterface, the implementation can differ and might not remove any data, depending on possibilities of the backend.

Options

`--cache-identifier`
: If set, this command only applies to the given cache

neos.flow:cache:flush

Flush all caches

The flush command flushes all caches (including code caches) which have been registered with Flow's Cache Manager. It will NOT remove any session data, unless you specifically configure the session caches to not be persistent.

If fatal errors caused by a package prevent the compile time bootstrap from running, the removal of any temporary data can be forced by specifying the option **–force**.

This command does not remove the precompiled data provided by frozen packages unless the **–force** option is used.

Options

`--force`
: Force flushing of any temporary data

Related commands

`neos.flow:cache:warmup`
: Warm up caches

`neos.flow:package:freeze`
: Freeze a package

`neos.flow:package:refreeze`
: Refreeze a package

neos.flow:cache:flushone

Flushes a particular cache by its identifier

Given a cache identifier, this flushes just that one cache. To find the cache identifiers, you can use the configuration:show command with the type set to "Caches".

Note that this does not have a force-flush option since it's not meant to remove temporary code data, resulting into a broken state if code files lack.

Arguments

`--identifier`
: Cache identifier to flush cache for

Related commands

`neos.flow:cache:flush`
: Flush all caches

`neos.flow:configuration:show`
: Show the active configuration settings

neos.flow:cache:list

List all configured caches and their status if available

This command will exit with a code 1 if at least one cache status contains errors or warnings This allows the command to be easily integrated in CI setups (the –quiet flag can be used to reduce verbosity)

Options

`--quiet`
: If set, this command only outputs errors & warnings

Related commands

`neos.flow:cache:show`
: Display details of a cache including a detailed status if available

neos.flow:cache:setup

Setup the given Cache if possible

Invokes the setup() method on the configured CacheBackend (if it implements the WithSetupInterface) which should setup and validate the backend (i.e. create required database tables, directories, ...)

Arguments

`--cache-identifier`

Related commands

`neos.flow:cache:list`
: List all configured caches and their status if available

`neos.flow:cache:setupall`
: Setup all Caches

neos.flow:cache:setupall

Setup all Caches

Invokes the setup() method on all configured CacheBackend that implement the WithSetupInterface interface which should setup and validate the backend (i.e. create required database tables, directories, ...)

This command will exit with a code 1 if at least one cache setup failed This allows the command to be easily integrated in CI setups (the –quiet flag can be used to reduce verbosity)

Options

`--quiet`
: If set, this command only outputs errors & warnings

Related commands

`neos.flow:cache:setup`
: Setup the given Cache if possible

neos.flow:cache:show

Display details of a cache including a detailed status if available

Arguments

`--cache-identifier`
: identifier of the cache (for example "Flow_Core")

Related commands

`neos.flow:cache:list`
: List all configured caches and their status if available

neos.flow:cache:warmup

Warm up caches

The warm up caches command initializes and fills – as far as possible – all registered caches to get a snappier response on the first following request. Apart from caches, other parts of the application may hook into this command and execute tasks which take further steps for preparing the app for the big rush.

Related commands

`neos.flow:cache:flush`
: Flush all caches

neos.flow:configuration:generateschema

Generate a schema for the given configuration or YAML file.

./flow configuration:generateschema –type Settings –path Neos.Flow.persistence

The schema will be output to standard output.

Options

`--type`
: Configuration type to create a schema for

`--path`
: path to the subconfiguration separated by "." like "Neos.Flow

`--yaml`
: YAML file to create a schema for

neos.flow:configuration:listtypes

List registered configuration types

neos.flow:configuration:show

Show the active configuration settings

The command shows the configuration of the current context as it is used by Flow itself. You can specify the configuration type and path if you want to show parts of the configuration.

Display all settings: ./flow configuration:show

Display Flow persistence settings: ./flow configuration:show –path Neos.Flow.persistence

Display Flow Object Cache configuration ./flow configuration:show –type Caches –path Flow_Object_Classes

Options

`--type`
: Configuration type to show, defaults to Settings

`--path`
: path to subconfiguration separated by "." like "Neos.Flow

neos.flow:configuration:validate

Validate the given configuration

Validate all configuration ./flow configuration:validate

Validate configuration at a certain subtype ./flow configuration:validate –type Settings –path Neos.Flow.persistence

You can retrieve the available configuration types with: ./flow configuration:listtypes

Options

`--type`
: Configuration type to validate

`--path`
: path to the subconfiguration separated by "." like "Neos.Flow

`--verbose`
: if true, output more verbose information on the schema files which were used

neos.flow:core:migrate

Migrate source files as needed

This will apply pending code migrations defined in packages to the specified package.

For every migration that has been run, it will create a commit in the package. This allows for easy inspection, rollback and use of the fixed code. If the affected package contains local changes or is not part of a git repository, the migration will be skipped. With the –force flag this behavior can be changed, but changes will only be committed if the working copy was clean before applying the migration.

Arguments

`--package`
: The key of the package to migrate

Options

`--status`
: Show the migration status, do not run migrations

`--packages-path`
: If set, use the given path as base when looking for packages

`--version`
: If set, execute only the migration with the given version (e.g. "20150119114100")

`--verbose`
: If set, notes and skipped migrations will be rendered

`--force`
: By default packages that are not under version control or contain local changes are skipped. With this flag set changes are applied anyways (changes are not committed if there are local changes though)

Related commands

neos.flow:doctrine:migrate
Migrate the database schema

neos.flow:core:setfilepermissions

Adjust file permissions for CLI and web server access

This command adjusts the file permissions of the whole Flow application to the given command line user and webserver user / group.

Arguments

--commandline-user
User name of the command line user, for example "john

--webserver-user
User name of the webserver, for example "www-data

--webserver-group
Group name of the webserver, for example "www-data

neos.flow:database:setcharset

Convert the database schema to use the given character set and collation (defaults to utf8mb4 and utf8mb4_unicode_ci).

This command can be used to convert the database configured in the Flow settings to the utf8mb4 character set and the utf8mb4_unicode_ci collation (by default, a custom collation can be given). It will only work when using the pdo_mysql driver.

Make a backup before using it, to be on the safe side. If you want to inspect the statements used for conversion, you can use the $output parameter to write them into a file. This file can be used to do the conversion manually.

For background information on this, see:

- http://stackoverflow.com/questions/766809/
- http://dev.mysql.com/doc/refman/5.5/en/alter-table.html
- https://medium.com/@adamhooper/in-mysql-never-use-utf8-use-utf8mb4-11761243e434
- https://mathiasbynens.be/notes/mysql-utf8mb4
- https://florian.ec/articles/mysql-doctrine-utf8/

The main purpose of this is to fix setups that were created with Flow before version 5.0. In those cases, the tables will have a collation that does not match the default collation of later Flow versions, potentially leading to problems when creating foreign key constraints (among others, potentially).

If you have special needs regarding the charset and collation, you *can* override the defaults with different ones.

Note: This command **is not a general purpose conversion tool**. It will specifically not fix cases of actual utf8 stored in latin1 columns. For this a conversion to BLOB followed by a conversion to the proper type, charset and collation is needed instead.

Options

--character-set
: Character set, defaults to utf8mb4

--collation
: Collation to use, defaults to utf8mb4_unicode_ci

--output
: A file to write SQL to, instead of executing it

--verbose
: If set, the statements will be shown as they are executed

neos.flow:doctrine:create

Create the database schema

Creates a new database schema based on the current mapping information.

It expects the database to be empty, if tables that are to be created already exist, this will lead to errors.

Options

--output
: A file to write SQL to, instead of executing it

Related commands

neos.flow:doctrine:update
: Update the database schema

neos.flow:doctrine:migrate
: Migrate the database schema

neos.flow:doctrine:dql

Run arbitrary DQL and display results

Any DQL queries passed after the parameters will be executed, the results will be output:

doctrine:dql –limit 10 'SELECT a FROM NeosFlowSecurityAccount a'

Options

--depth
: How many levels deep the result should be dumped

--hydration-mode
: One of: object, array, scalar, single-scalar, simpleobject

--offset
: Offset the result by this number

`--limit`
: Limit the result to this number

neos.flow:doctrine:entitystatus

Show the current status of entities and mappings

Shows basic information about which entities exist and possibly if their mapping information contains errors or not.

To run a full validation, use the validate command.

Options

`--dump-mapping-data`
: If set, the mapping data will be output

`--entity-class-name`
: If given, the mapping data for just this class will be output

Related commands

`neos.flow:doctrine:validate`
: Validate the class/table mappings

neos.flow:doctrine:migrate

Migrate the database schema

Adjusts the database structure by applying the pending migrations provided by currently active packages.

Options

`--version`
: The version to migrate to

`--output`
: A file to write SQL to, instead of executing it

`--dry-run`
: Whether to do a dry run or not

`--quiet`
: If set, only the executed migration versions will be output, one per line

Related commands

neos.flow:doctrine:migrationstatus
Show the current migration status

neos.flow:doctrine:migrationexecute
Execute a single migration

neos.flow:doctrine:migrationgenerate
Generate a new migration

neos.flow:doctrine:migrationversion
Mark/unmark migrations as migrated

neos.flow:doctrine:migrationexecute

Execute a single migration

Manually runs a single migration in the given direction.

Arguments

--version
The migration to execute

Options

--direction
Whether to execute the migration up (default) or down

--output
A file to write SQL to, instead of executing it

--dry-run
Whether to do a dry run or not

Related commands

neos.flow:doctrine:migrate
Migrate the database schema

neos.flow:doctrine:migrationstatus
Show the current migration status

neos.flow:doctrine:migrationgenerate
Generate a new migration

neos.flow:doctrine:migrationversion
Mark/unmark migrations as migrated

neos.flow:doctrine:migrationgenerate

Generate a new migration

If $diffAgainstCurrent is true (the default), it generates a migration file with the diff between current DB structure and the found mapping metadata.

Otherwise an empty migration skeleton is generated.

Only includes tables/sequences matching the $filterExpression regexp when diffing models and existing schema. Include delimiters in the expression! The use of

–filter-expression '/^acme_com/'

would only create a migration touching tables starting with "acme_com".

Note: A filter-expression will overrule any filter configured through the Neos.Flow.persistence.doctrine.migrations.ignoredTables setting

Options

`--diff-against-current`
: Whether to base the migration on the current schema structure

`--filter-expression`
: Only include tables/sequences matching the filter expression regexp

`--force`
: Generate migrations even if there are migrations left to execute

Related commands

`neos.flow:doctrine:migrate`
: Migrate the database schema

`neos.flow:doctrine:migrationstatus`
: Show the current migration status

`neos.flow:doctrine:migrationexecute`
: Execute a single migration

`neos.flow:doctrine:migrationversion`
: Mark/unmark migrations as migrated

neos.flow:doctrine:migrationstatus

Show the current migration status

Displays the migration configuration as well as the number of available, executed and pending migrations.

Options

`--show-migrations`
: Output a list of all migrations and their status

Related commands

`neos.flow:doctrine:migrate`
: Migrate the database schema

`neos.flow:doctrine:migrationexecute`
: Execute a single migration

`neos.flow:doctrine:migrationgenerate`
: Generate a new migration

`neos.flow:doctrine:migrationversion`
: Mark/unmark migrations as migrated

neos.flow:doctrine:migrationversion

Mark/unmark migrations as migrated

If *all* is given as version, all available migrations are marked as requested.

Arguments

`--version`
: The migration to execute

Options

`--add`
: The migration to mark as migrated

`--delete`
: The migration to mark as not migrated

Related commands

`neos.flow:doctrine:migrate`
: Migrate the database schema

`neos.flow:doctrine:migrationstatus`
: Show the current migration status

`neos.flow:doctrine:migrationexecute`
: Execute a single migration

`neos.flow:doctrine:migrationgenerate`
: Generate a new migration

neos.flow:doctrine:update

Update the database schema

Updates the database schema without using existing migrations.

It will not drop foreign keys, sequences and tables, unless *–unsafe-mode* is set.

Options

`--unsafe-mode`
: If set, foreign keys, sequences and tables can potentially be dropped.

`--output`
: A file to write SQL to, instead of executing the update directly

Related commands

`neos.flow:doctrine:create`
: Create the database schema

`neos.flow:doctrine:migrate`
: Migrate the database schema

neos.flow:doctrine:validate

Validate the class/table mappings

Checks if the current class model schema is valid. Any inconsistencies in the relations between models (for example caused by wrong or missing annotations) will be reported.

Note that this does not check the table structure in the database in any way.

Related commands

`neos.flow:doctrine:entitystatus`
: Show the current status of entities and mappings

neos.flow:help:help

Display help for a command

The help command displays help for a given command: ./flow help <commandIdentifier>

Options

`--command-identifier`
: Identifier of a command for more details

neos.flow:middleware:list

Lists all configured middleware components in the order they will be executed

neos.flow:package:create

Create a new package

This command creates a new package which contains only the mandatory directories and files.

Arguments

`--package-key`
: The package key of the package to create

Options

`--package-type`
: The package type of the package to create

Related commands

`neos.kickstarter:kickstart:package`
: Kickstart a new package

neos.flow:package:freeze

Freeze a package

This function marks a package as **frozen** in order to improve performance in a development context. While a package is frozen, any modification of files within that package won't be tracked and can lead to unexpected behavior.

File monitoring won't consider the given package. Further more, reflection data for classes contained in the package is cached persistently and loaded directly on the first request after caches have been flushed. The precompiled reflection data is stored in the **Configuration** directory of the respective package.

By specifying **all** as a package key, all currently frozen packages are frozen (the default).

Options

`--package-key`
: Key of the package to freeze

Related commands

`neos.flow:package:unfreeze`
: Unfreeze a package

`neos.flow:package:refreeze`
: Refreeze a package

neos.flow:package:list

List available packages

Lists all locally available packages. Displays the package key, version and package title.

Options

`--loading-order`
: The returned packages are ordered by their loading order.

neos.flow:package:refreeze

Refreeze a package

Refreezes a currently frozen package: all precompiled information is removed and file monitoring will consider the package exactly once, on the next request. After that request, the package remains frozen again, just with the updated data.

By specifying **all** as a package key, all currently frozen packages are refrozen (the default).

Options

`--package-key`
: Key of the package to refreeze, or 'all'

Related commands

`neos.flow:package:freeze`
: Freeze a package

`neos.flow:cache:flush`
: Flush all caches

neos.flow:package:rescan

Rescan package availability and recreates the PackageStates configuration.

neos.flow:package:unfreeze

Unfreeze a package

Unfreezes a previously frozen package. On the next request, this package will be considered again by the file monitoring and related services – if they are enabled in the current context.

By specifying **all** as a package key, all currently frozen packages are unfrozen (the default).

Options

--package-key
: Key of the package to unfreeze, or 'all'

Related commands

neos.flow:package:freeze
: Freeze a package

neos.flow:cache:flush
: Flush all caches

neos.flow:resource:clean

Clean up resource registry

This command checks the resource registry (that is the database tables) for orphaned resource objects which don't seem to have any corresponding data anymore (for example: the file in Data/Persistent/Resources has been deleted without removing the related PersistentResource object).

If the Neos.Media package is active, this command will also detect any assets referring to broken resources and will remove the respective Asset object from the database when the broken resource is removed.

This command will ask you interactively what to do before deleting anything.

neos.flow:resource:copy

Copy resources

This command copies all resources from one collection to another storage identified by name. The target storage must be empty and must not be identical to the current storage of the collection.

This command merely copies the binary data from one storage to another, it does not change the related PersistentResource objects in the database in any way. Since the PersistentResource objects in the database refer to a collection name, you can use this command for migrating from one storage to another my configuring the new storage with the name of the old storage collection after the resources have been copied.

Arguments

`--source-collection`
: The name of the collection you want to copy the assets from

`--target-collection`
: The name of the collection you want to copy the assets to

Options

`--publish`
: If enabled, the target collection will be published after the resources have been copied

neos.flow:resource:publish

Publish resources

This command publishes the resources of the given or - if none was specified, all - resource collections to their respective configured publishing targets.

Options

`--collection`
: If specified, only resources of this collection are published. Example: 'persistent'

neos.flow:routing:list

List the known routes

This command displays a list of all currently registered routes.

neos.flow:routing:match

Match the given URI to a corresponding route

This command takes an incoming URI and displays the matched Route and the mapped routing values (if any):

./flow routing:match "/de" –parameters="{"requestUriHost": "localhost"}"

Arguments

`--uri`
: The incoming route, absolute or relative

Options

--method
: The HTTP method to simulate (default is 'GET')

--parameters
: Route parameters as JSON string. Make sure to specify this option as described in the description in order to prevent parsing issues

neos.flow:routing:resolve

Build an URI for the given parameters

This command takes package, controller and action and displays the resolved URI and which route matched (if any):

./flow routing:resolve Some.Package –controller SomeController –additional-arguments="{"some-argument": "some-value"}"

Arguments

--package
: Package key (according to "@package" route value)

Options

--controller
: Controller name (according to "@controller" route value), default is 'Standard'

--action
: Action name (according to "@action" route value), default is 'index'

--format
: Requested Format name (according to "@format" route value), default is 'html'

--subpackage
: SubPackage name (according to "@subpackage" route value)

--additional-arguments
: Additional route values as JSON string. Make sure to specify this option as described in the description in order to prevent parsing issues

--parameters
: Route parameters as JSON string. Make sure to specify this option as described in the description in order to prevent parsing issues

--base-uri
: Base URI of the simulated request, default ist 'http://localhost'

--force-absolute-uri
: Whether or not to force the creation of an absolute URI

neos.flow:routing:show

Show information for a route

This command displays the configuration of a route specified by index number.

Arguments

`--index`
: The index of the route as given by routing:list

neos.flow:schema:validate

Validate the given configurationfile againt a schema file

Options

`--configuration-file`
: path to the validated configuration file

`--schema-file`
: path to the schema file

`--verbose`
: if true, output more verbose information on the schema files which were used

neos.flow:security:describerole

Show details of a specified role

Arguments

`--role`
: identifier of the role to describe (for example "Neos.Flow:Everybody")

neos.flow:security:generatekeypair

Generate a public/private key pair and add it to the RSAWalletService

Options

`--used-for-passwords`
: If the private key should be used for passwords

Related commands

neos.flow:security:importprivatekey
Import a private key

neos.flow:security:importprivatekey

Import a private key

Read a PEM formatted private key from stdin and import it into the RSAWalletService. The public key will be automatically extracted and stored together with the private key as a key pair.

You can generate the same fingerprint returned from this using these commands:

ssh-keygen -yf my-key.pem > my-key.pub ssh-keygen -lf my-key.pub

To create a private key to import using this method, you can use:

ssh-keygen -t rsa -f my-key ./flow security:importprivatekey < my-key

Again, the fingerprint can also be generated using:

ssh-keygen -lf my-key.pub

Options

--used-for-passwords
If the private key should be used for passwords

Related commands

neos.flow:security:importpublickey
Import a public key

neos.flow:security:generatekeypair
Generate a public/private key pair and add it to the RSAWalletService

neos.flow:security:importpublickey

Import a public key

Read a PEM formatted public key from stdin and import it into the RSAWalletService.

Related commands

neos.flow:security:importprivatekey
Import a private key

neos.flow:security:listroles

List all configured roles

Options

`--include-abstract`
: Set this flag to include abstract roles

neos.flow:security:showeffectivepolicy

Shows a list of all defined privilege targets and the effective permissions

Arguments

`--privilege-type`
: The privilege type ("entity", "method" or the FQN of a class implementing PrivilegeInterface)

Options

`--roles`
: A comma separated list of role identifiers. Shows policy for an unauthenticated user when left empty.

neos.flow:security:showmethodsforprivilegetarget

Shows the methods represented by the given security privilege target

If the privilege target has parameters those can be specified separated by a colon for example "parameter1:value1" "parameter2:value2". But be aware that this only works for parameters that have been specified in the policy

Arguments

`--privilege-target`
: The name of the privilegeTarget as stated in the policy

neos.flow:security:showunprotectedactions

Lists all public controller actions not covered by the active security policy

neos.flow:server:run

Run a standalone development server

Starts an embedded server, see http://php.net/manual/en/features.commandline.webserver.php Note: This requires PHP 5.4+

To change the context Flow will run in, you can set the **FLOW_CONTEXT** environment variable: *export FLOW_CONTEXT=Development && ./flow server:run*

Options

`--host`
: The host name or IP address for the server to listen on

`--port`
: The server port to listen on

neos.flow:session:collectgarbage

Run garbage collection for sesions.

This command will remove session-data and -metadate of outdated sessions identified by lastActivityTimestamp being older than inactivityTimeout

!!! This is usually done automatically after shutdown for the percentage of requests specified in the setting *Neos.Flow.session.garbageCollection.probability*

Use this command if you need more direct control over the cleanup intervals.

neos.flow:session:destroyall

Destroys all sessions.

This special command is needed, because sessions are kept in persistent storage and are not flushed with other caches by default.

This is functionally equivalent to *./flow flow:cache:flushOne Flow_Session_Storage && ./flow flow:cache:flushOne Flow_Session_MetaData*

neos.flow:signal:listconnected

Lists all connected signals with their slots.

Options

`--class-name`
: if specified, only signals matching the given fully qualified class name will be shown. Note: escape namespace separators or wrap the value in quotes, e.g. "–class-name Neos\Flow\Core\Bootstrap".

`--method-name`
: if specified, only signals matching the given method name will be shown. This is only useful in conjunction with the "–class-name" option.

`neos.flow:typeconverter:list`

Lists all currently active and registered type converters

All active converters are listed with ordered by priority and grouped by source type first and target type second.

Options

`--source`
: Filter by source

`--target`
: Filter by target type

1.8.3 Package *NEOS.FLUIDADAPTOR*

`neos.fluidadaptor:documentation:generatexsd`

Generate Fluid ViewHelper XSD Schema

Generates Schema documentation (XSD) for your ViewHelpers, preparing the file to be placed online and used by any XSD-aware editor. After creating the XSD file, reference it in your IDE and import the namespace in your Fluid template by adding the xmlns:* attribute(s): <html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="https://neos.io/ns/Neos/Neos/ViewHelpers" ...>

Arguments

`--php-namespace`
: Namespace of the Fluid ViewHelpers without leading backslash (for example 'NeosFluidAdaptorViewHelpers'). NOTE: Quote and/or escape this argument as needed to avoid backslashes from being interpreted!

Options

`--xsd-namespace`
: Unique target namespace used in the XSD schema (for example "http://yourdomain.org/ns/viewhelpers"). Defaults to "https://neos.io/ns/<php namespace>".

`--target-file`
: File path and name of the generated XSD schema. If not specified the schema will be output to standard output.

`--xsd-domain`
: Domain used in the XSD schema (for example "http://yourdomain.org"). Defaults to "https://neos.io".

1.8.4 Package *NEOS.KICKSTARTER*

`neos.kickstarter:kickstart:actioncontroller`

Kickstart a new action controller

Generates an Action Controller with the given name in the specified package. In its default mode it will create just the controller containing a sample indexAction.

By specifying the –generate-actions flag, this command will also create a set of actions. If no model or repository exists which matches the controller name (for example "CoffeeRepository" for "CoffeeController"), an error will be shown.

Likewise the command exits with an error if the specified package does not exist. By using the –generate-related flag, a missing package, model or repository can be created alongside, avoiding such an error.

By specifying the –generate-templates flag, this command will also create matching Fluid templates for the actions created. Alternatively, by specifying the –generate-fusion flag, this command will create matching Fusion files for the actions.

The default behavior is to not overwrite any existing code. This can be overridden by specifying the –force flag.

Arguments

`--package-key`
: The package key of the package for the new controller with an optional subpackage, (e.g. "MyCompany.MyPackage/Admin").

`--controller-name`
: The name for the new controller. This may also be a comma separated list of controller names.

Options

`--generate-actions`
: Also generate index, show, new, create, edit, update and delete actions.

`--generate-templates`
: Also generate the templates for each action.

`--generate-fusion`
: If Fusion templates should be generated instead of Fluid.

`--generate-related`
: Also create the mentioned package, related model and repository if necessary.

`--force`
: Overwrite any existing controller or template code. Regardless of this flag, the package, model and repository will never be overwritten.

Related commands

neos.kickstarter:kickstart:commandcontroller
Kickstart a new command controller

neos.kickstarter:kickstart:commandcontroller

Kickstart a new command controller

Creates a new command controller with the given name in the specified package. The generated controller class already contains an example command.

Arguments

--package-key
The package key of the package for the new controller

--controller-name
The name for the new controller. This may also be a comma separated list of controller names.

Options

--force
Overwrite any existing controller.

Related commands

neos.kickstarter:kickstart:actioncontroller
Kickstart a new action controller

neos.kickstarter:kickstart:documentation

Kickstart documentation

Generates a documentation skeleton for the given package.

Arguments

--package-key
The package key of the package for the documentation

neos.kickstarter:kickstart:model

Kickstart a new domain model

This command generates a new domain model class. The fields are specified as a variable list of arguments with field name and type separated by a colon (for example "title:string" "size:int" "type:MyType").

Arguments

--package-key
The package key of the package for the domain model

--model-name
The name of the new domain model class

Options

--force
Overwrite any existing model.

Related commands

neos.kickstarter:kickstart:repository
Kickstart a new domain repository

neos.kickstarter:kickstart:package

Kickstart a new package

Creates a new package and creates a standard Action Controller and a sample template for its Index Action.

For creating a new package without sample code use the package:create command.

Arguments

--package-key
The package key, for example "MyCompany.MyPackageName

Options

--package-type
Optional package type, e.g. "neos-plugin

Related commands

neos.flow:package:create
Create a new package

neos.kickstarter:kickstart:repository

Kickstart a new domain repository

This command generates a new domain repository class for the given model name.

Arguments

--package-key
The package key

--model-name
The name of the domain model class

Options

--force
Overwrite any existing repository.

Related commands

neos.kickstarter:kickstart:model
Kickstart a new domain model

neos.kickstarter:kickstart:translation

Kickstart translation

Generates the translation files for the given package.

Arguments

--package-key
The package key of the package for the translation

--source-language-key
The language key of the default language

Options

`--target-language-keys`
: Comma separated language keys for the target translations

1.8.5 Package *NEOS.MEDIA*

`neos.media:media:clearthumbnails`

Remove thumbnails

Removes all thumbnail objects and their resources. Optional `preset` parameter to only remove thumbnails matching a specific thumbnail preset configuration.

Options

`--preset`
: Preset name, if provided only thumbnails matching that preset are cleared

`--quiet`
: If set, only errors will be displayed.

`neos.media:media:createthumbnails`

Create thumbnails

Creates thumbnail images based on the configured thumbnail presets. Optional `preset` parameter to only create thumbnails for a specific thumbnail preset configuration.

Additionally, accepts a `async` parameter determining if the created thumbnails are generated when created.

Options

`--preset`
: Preset name, if not provided thumbnails are created for all presets

`--async`
: Asynchronous generation, if not provided the setting `Neos.Media.asyncThumbnails` is used

`--quiet`
: If set, only errors will be displayed.

`neos.media:media:importresources`

Import resources to asset management

This command detects Flow "PersistentResource"s which are not yet available as "Asset" objects and thus don't appear in the asset management. The type of the imported asset is determined by the file extension provided by the PersistentResource.

Options

`--simulate`
: If set, this command will only tell what it would do instead of doing it right away

`--quiet`

neos.media:media:listvariantpresets

List all configurations for your imageVariants.

Doesn't matter if configured under 'Neos.Media.variantPresets' or already deleted from this configuration. This command will find every single one for you.

neos.media:media:removeunused

Remove unused assets

This command iterates over all existing assets, checks their usage count and lists the assets which are not reported as used by any AssetUsageStrategies. The unused assets can than be removed.

Options

`--asset-source`
: If specified, only assets of this asset source are considered. For example "neos" or "my-asset-management-system

`--quiet`
: If set, only errors will be displayed.

`--assume-yes`
: If set, "yes" is assumed for the "shall I remove …" dialogs

`--only-tags`
: Comma-separated list of asset tag labels, that should be taken into account

`--limit`
: Limit the result of unused assets displayed and removed for this run.

`--only-collections`
: Comma-separated list of asset collection titles, that should be taken into account

neos.media:media:removevariants

Cleanup imageVariants with provided identifier and variant name.

Image variants that are still configured are removed without usage check and can be regenerated afterwards with *media:renderVariants*.

This command will not remove any custom cropped image variants.

Arguments

--identifier
Identifier of variants to remove.

--variant-name
Variants with this name will be removed (if exist).

Options

--quiet
If set, only errors and questions will be displayed.

--assume-yes
If set, "yes" is assumed for the "shall I remove …" dialog.

--limit
Limit the result of unused assets displayed and removed for this run.

neos.media:media:renderthumbnails

Render ungenerated thumbnails

Loops over ungenerated thumbnails and renders them. Optional `limit` parameter to limit the amount of thumbnails to be rendered to avoid memory exhaustion.

Options

--limit
Limit the amount of thumbnails to be rendered to avoid memory exhaustion

--quiet
If set, only errors will be displayed.

neos.media:media:rendervariants

Render asset variants

Loops over missing configured asset variants and renders them. Optional `limit` parameter to limit the amount of variants to be rendered to avoid memory exhaustion.

If the re-render parameter is given, any existing variants will be rendered again, too.

Options

--limit
Limit the amount of variants to be rendered to avoid memory exhaustion

--quiet
If set, only errors will be displayed.

--recreate
If set, existing asset variants will be re-generated and replaced

1.8.6 Package *NEOS.NEOS*

`neos.neos:domain:activate`

Activate a domain record by hostname (with globbing)

Arguments

`--hostname`
: The hostname to activate (globbing is supported)

`neos.neos:domain:add`

Add a domain record

Arguments

`--site-node-name`
: The nodeName of the site rootNode, e.g. “flowneosio

`--hostname`
: The hostname to match on, e.g. “flow.neos.io

Options

`--scheme`
: The scheme for linking (http/https)

`--port`
: The port for linking (0-49151)

`neos.neos:domain:deactivate`

Deactivate a domain record by hostname (with globbing)

Arguments

`--hostname`
: The hostname to deactivate (globbing is supported)

neos.neos:domain:delete

Delete a domain record by hostname (with globbing)

Arguments

`--hostname`
: The hostname to remove (globbing is supported)

neos.neos:domain:list

Display a list of available domain records

Options

`--hostname`
: An optional hostname to search for

neos.neos:site:activate

Activate a site (with globbing)

This command activates the specified site.

Arguments

`--site-node`
: The node name of the sites to activate (globbing is supported)

neos.neos:site:create

Create a new site

This command allows to create a blank site with just a single empty document in the default dimension. The name of the site, the packageKey must be specified.

The node type given with the `nodeType` option must already exists and have the superType `Neos.Neos:Document`.

If no `nodeName` option is specified the command will create a unique node-name from the name of the site. If a node name is given it has to be unique for the setup.

If the flag `activate` is set to false new site will not be activated.

Arguments

--name
The name of the site

--package-key
The site package

--node-type
The node type to use for the site node, e.g. Amce.Com:Page

Options

--node-name
The name of the site node. If no nodeName is given it will be determined from the siteName.

--inactive
The new site is not activated immediately (default = false)

neos.neos:site:deactivate

Deactivate a site (with globbing)

This command deactivates the specified site.

Arguments

--site-node
The node name of the sites to deactivate (globbing is supported)

neos.neos:site:export

Export sites content (e.g. site:export –package-key "Neos.Demo")

This command exports all or one specific site with all its content into an XML format.

If the package key option is given, the site(s) will be exported to the given package in the default location Resources/Private/Content/Sites.xml.

If the filename option is given, any resources will be exported to files in a folder named "Resources" alongside the XML file.

If neither the filename nor the package key option are given, the XML will be printed to standard output and assets will be embedded into the XML in base64 encoded form.

Options

--site-node
the node name of the site to be exported; if none given will export all sites

--tidy
Whether to export formatted XML. This is defaults to true

--filename
relative path and filename to the XML file to create. Any resource will be stored in a sub folder "Resources".

--package-key
Package to store the XML file in. Any resource will be stored in a sub folder "Resources".

--node-type-filter
Filter the node type of the nodes, allows complex expressions (e.g. "Neos.Neos:Page", "!Neos.Neos:Page,Neos.Neos:Text")

neos.neos:site:import

Import sites content

This command allows for importing one or more sites or partial content from an XML source. The format must be identical to that produced by the export command.

If a filename is specified, this command expects the corresponding file to contain the XML structure. The filename php://stdin can be used to read from standard input.

If a package key is specified, this command expects a Sites.xml file to be located in the private resources directory of the given package (Resources/Private/Content/Sites.xml).

Options

--package-key
Package key specifying the package containing the sites content

--filename
relative path and filename to the XML file containing the sites content

neos.neos:site:list

List available sites

neos.neos:site:prune

Remove site with content and related data (with globbing)

In the future we need some more sophisticated cleanup.

Arguments

`--site-node`
: Name for site root nodes to clear only content of this sites (globbing is supported)

neos.neos:user:activate

Activate a user (with globbing)

This command reactivates possibly expired accounts for the given user.

If an authentication provider is specified, this command will look for an account with the given username related to the given provider. Still, this command will activate **all** accounts of a user, once such a user has been found.

Arguments

`--username`
: The username of the user to be activated (globbing is supported)

Options

`--authentication-provider`
: Name of the authentication provider to use for finding the user. Example: "Neos.Neos:Backend

neos.neos:user:addrole

Add a role to a user

This command allows for adding a specific role to an existing user.

Roles can optionally be specified as a comma separated list. For all roles provided by Neos, the role namespace "Neos.Neos:" can be omitted.

If an authentication provider was specified, the user will be determined by an account identified by "username" related to the given provider. However, once a user has been found, the new role will be added to **all** existing accounts related to that user, regardless of its authentication provider.

Arguments

`--username`
: The username of the user (globbing is supported)

`--role`
: Role to be added to the user, for example "Neos.Neos:Administrator" or just "Administrator

Options

`--authentication-provider`
: Name of the authentication provider to use. Example: "Neos.Neos:Backend

neos.neos:user:create

Create a new user

This command creates a new user which has access to the backend user interface.

More specifically, this command will create a new user and a new account at the same time. The created account is, by default, a Neos backend account using the the "Neos.Neos:Backend" for authentication. The given username will be used as an account identifier for that new account.

If an authentication provider name is specified, the new account will be created for that provider instead.

Roles for the new user can optionally be specified as a comma separated list. For all roles provided by Neos, the role namespace "Neos.Neos:" can be omitted.

Arguments

`--username`
: The username of the user to be created, used as an account identifier for the newly created account

`--password`
: Password of the user to be created

`--first-name`
: First name of the user to be created

`--last-name`
: Last name of the user to be created

Options

`--roles`
: A comma separated list of roles to assign. Examples: "Editor, Acme.Foo:Reviewer

`--authentication-provider`
: Name of the authentication provider to use for the new account. Example: "Neos.Neos:Backend

neos.neos:user:deactivate

Deactivate a user (with globbing)

This command deactivates a user by flagging all of its accounts as expired.

If an authentication provider is specified, this command will look for an account with the given username related to the given provider. Still, this command will deactivate **all** accounts of a user, once such a user has been found.

Arguments

`--username`
: The username of the user to be deactivated (globbing is supported)

Options

`--authentication-provider`
: Name of the authentication provider to use for finding the user. Example: "Neos.Neos:Backend

neos.neos:user:delete

Delete a user (with globbing)

This command deletes an existing Neos user. All content and data directly related to this user, including but not limited to draft workspace contents, will be removed as well.

All accounts owned by the given user will be deleted.

If an authentication provider is specified, this command will look for an account with the given username related to the given provider. Specifying an authentication provider does **not** mean that only the account for that provider is deleted! If a user was found by the combination of username and authentication provider, **all** related accounts will be deleted.

Arguments

`--username`
: The username of the user to be removed (globbing is supported)

Options

`--assume-yes`
: Assume "yes" as the answer to the confirmation dialog

`--authentication-provider`
: Name of the authentication provider to use. Example: "Neos.Neos:Backend

neos.neos:user:list

List all users

This command lists all existing Neos users.

neos.neos:user:removerole

Remove a role from a user

This command allows for removal of a specific role from an existing user.

If an authentication provider was specified, the user will be determined by an account identified by "username" related to the given provider. However, once a user has been found, the role will be removed from **all** existing accounts related to that user, regardless of its authentication provider.

Arguments

`--username`
: The username of the user (globbing is supported)

`--role`
: Role to be removed from the user, for example "Neos.Neos:Administrator" or just "Administrator

Options

`--authentication-provider`
: Name of the authentication provider to use. Example: "Neos.Neos:Backend

neos.neos:user:setpassword

Set a new password for the given user

This command sets a new password for an existing user. More specifically, all accounts related to the user which are based on a username / password token will receive the new password.

If an authentication provider was specified, the user will be determined by an account identified by "username" related to the given provider.

Arguments

`--username`
: Username of the user to modify

`--password`
: The new password

Options

`--authentication-provider`
: Name of the authentication provider to use for finding the user. Example: "Neos.Neos:Backend

neos.neos:user:show

Shows the given user

This command shows some basic details about the given user. If such a user does not exist, this command will exit with a non-zero status code.

The user will be retrieved by looking for a Neos backend account with the given identifier (ie. the username) and then retrieving the user which owns that account. If an authentication provider is specified, this command will look for an account identified by "username" for that specific provider.

Arguments

`--username`
: The username of the user to show. Usually refers to the account identifier of the user's Neos backend account.

Options

`--authentication-provider`
: Name of the authentication provider to use. Example: "Neos.Neos:Backend

neos.neos:workspace:create

Create a new workspace

This command creates a new workspace.

Arguments

`--workspace`
: Name of the workspace, for example "christmas-campaign

Options

`--base-workspace`
: Name of the base workspace. If none is specified, "live" is assumed.

`--title`
: Human friendly title of the workspace, for example "Christmas Campaign

`--description`
: A description explaining the purpose of the new workspace

`--owner`
: The identifier of a User to own the workspace

neos.neos:workspace:delete

Deletes a workspace

This command deletes a workspace. If you only want to empty a workspace and not delete the workspace itself, use *workspace:discard* instead.

Arguments

`--workspace`
: Name of the workspace, for example "christmas-campaign

Options

`--force`
: Delete the workspace and all of its contents

Related commands

`neos.neos:workspace:discard`
: Discard changes in workspace

neos.neos:workspace:discard

Discard changes in workspace

This command discards all modified, created or deleted nodes in the specified workspace.

Arguments

`--workspace`
: Name of the workspace, for example "user-john

Options

`--verbose`
: If enabled, information about individual nodes will be displayed

`--dry-run`
: If set, only displays which nodes would be discarded, no real changes are committed

neos.neos:workspace:list

Display a list of existing workspaces

neos.neos:workspace:publish

Publish changes of a workspace

This command publishes all modified, created or deleted nodes in the specified workspace to its base workspace. If a target workspace is specified, the content is published to that workspace instead.

Arguments

`--workspace`
: Name of the workspace containing the changes to publish, for example "user-john

Options

`--target-workspace`
: If specified, the content will be published to this workspace instead of the base workspace

`--verbose`
: If enabled, some information about individual nodes will be displayed

`--dry-run`
: If set, only displays which nodes would be published, no real changes are committed

neos.neos:workspace:rebase

Rebase a workspace

This command sets a new base workspace for the specified workspace. Note that doing so will put the possible changes contained in the workspace to be rebased into a different context and thus might lead to unintended results when being published.

Arguments

`--workspace`
: Name of the workspace to rebase, for example "user-john

`--base-workspace`
: Name of the new base workspace

1.8.7 Package *NEOS.SITEKICKSTARTER*

`neos.sitekickstarter:kickstart:site`

Kickstart a new site package

This command generates a new site package with basic Fusion and Sites.xml

Arguments

`--package-key`
: The packageKey for your site

`--site-name`
: The siteName of your site

1.9 Validator Reference

1.9.1 Flow Validator Reference

This reference was automatically generated from code on 2024-03-10

AggregateBoundaryValidator

A validator which will not validate Aggregates that are lazy loaded and uninitialized. Validation over Aggregate Boundaries can hence be forced by making the relation to other Aggregate Roots eager loaded.

Note that this validator is not part of the public API and you should not use it manually.

Checks if the given value is valid according to the validator, and returns the Error Messages object which occurred. Will skip validation if value is an uninitialized lazy loading proxy.

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `skipUnInitializedProxies` (boolean, *optional*): Whether proxies not yet initialized should be skipped during validation

AlphanumericValidator

Validator for alphanumeric strings.

The given $value is valid if it is an alphanumeric string, which is defined as [[:alnum:]].

Note: A value of NULL or an empty string (‘’) is considered valid

---

BooleanValueValidator

Validator for a specific boolean value.

Checks if the given value is a specific boolean value.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `expectedValue` (boolean, *optional*): The expected boolean value

CollectionValidator

A generic collection validator.

Checks for a collection and if needed validates the items in the collection. This is done with the specified element validator or a validator based on the given element type and validation group.

Either elementValidator or elementType must be given, otherwise validation will be skipped.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `elementValidator` (string, *optional*): The validator type to use for the collection elements
- `elementValidatorOptions` (array, *optional*): The validator options to use for the collection elements
- `elementType` (string, *optional*): The type of the elements in the collection
- `validationGroups` (string, *optional*): The validation groups to link to

CountValidator

Validator for countable things

The given value is valid if it is an array or Countable that contains the specified amount of elements.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `minimum` (integer, *optional*): The minimum count to accept
- `maximum` (integer, *optional*): The maximum count to accept

DateTimeRangeValidator

Validator for checking Date and Time boundaries

Adds errors if the given DateTime does not match the set boundaries.

latestDate and earliestDate may be each <time>, <start>/<duration> or <duration>/<end>, where <duration> is an ISO 8601 duration and <start> or <end> or <time> may be 'now' or a PHP supported format. (1)

In general, you are able to provide a timestamp or a timestamp with additional calculation. Calculations are done as described in ISO 8601 (2), with an introducing "P". P7MT2H30M for example mean a period of 7 months, 2 hours and 30 minutes (P introduces a period at all, while a following T introduces the time-section of a period. This is not at least in order not to confuse months and minutes, both represented as M). A period is separated from the timestamp with a forward slash "/". If the period follows the timestamp, that period is added to the timestamp; if the period precedes the timestamp, it's subtracted. The timestamp can be one of PHP's supported date formats (1), so also "now" is supported.

Use cases:

If you offer something that has to be manufactured and you ask for a delivery date, you might assure that this date is at least two weeks in advance; this could be done with the expression "now/P2W". If you have a library of ancient goods and want to track a production date that is at least 5 years ago, you can express it with "P5Y/now".

Examples:

If you want to test if a given date is at least five minutes ahead, use
earliestDate: now/PT5M

If you want to test if a given date was at least 10 days ago, use
latestDate: P10D/now

If you want to test if a given date is between two fix boundaries, just combine the latestDate and earliestDate-options:
earliestDate: 2007-03-01T13:00:00Z latestDate: 2007-03-30T13:00:00Z

Footnotes:

http://de.php.net/manual/en/datetime.formats.compound.php (1) http://en.wikipedia.org/wiki/ISO_8601#Durations (2) http://en.wikipedia.org/wiki/ISO_8601#Time_intervals (3)

Note: A value of NULL or an empty string ('') is considered valid

---

Arguments

- `latestDate` (string, *optional*): The latest date to accept
- `earliestDate` (string, *optional*): The earliest date to accept

DateTimeValidator

Validator for DateTime objects.

Checks if the given value is a valid DateTime object.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `locale` (string|Locale, *optional*): The locale to use for date parsing
- `strictMode` (boolean, *optional*): Use strict mode for date parsing
- `formatLength` (string, *optional*): The format length, see DatesReader::FORMAT_LENGTH_*
- `formatType` (string, *optional*): The format type, see DatesReader::FORMAT_TYPE_*

EmailAddressValidator

Validator for email addresses

Checks if the given value is a valid email address.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `strict` (bool, *optional*): Whether to fail validation on RFC warnings
- `checkDns` (bool, *optional*): Whether to use DNS checks

FileExtensionValidator

The given $value is has one of the allowed file extensions Note: a value of NULL or empty string (‘’) are considered valid

The given $value is valid media type matches one of the allowedTypes and none of the disallowedTypes

Note: a value of NULL or empty string (‘’) is considered valid and was handled already

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `allowedExtensions` (array): Array of allowed file extensions

FileSizeValidator

Validator for file sizes Note: a value of NULL or empty string (‘’) are considered valid

The given $value is valid media type matches one of the allowedTypes and none of the disallowedTypes

Note: a value of NULL or empty string (‘’) is considered valid and was handled already

---

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `minimum` (integer): Minimum allowed filesize in bytes
- `maximum` (integer): Maximum allowed filesize in bytes

FloatValidator

Validator for floats.

The given value is valid if it is of type float or a string matching the regular expression [0-9.e+-]

Note: A value of NULL or an empty string (‘’) is considered valid

---

GenericObjectValidator

A generic object validator which allows for specifying property validators.

Checks if the given value is valid according to the property validators.

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `skipUnInitializedProxies` (boolean, *optional*): Whether proxies not yet initialized should be skipped during validation

IntegerValidator

Validator for integers.

Checks if the given value is a valid integer.

Note: A value of NULL or an empty string (‘’) is considered valid

---

LabelValidator

A validator for labels.

Labels usually allow all kinds of letters, numbers, punctuation marks and the space character. What you don’t want in labels though are tabs, new line characters or HTML tags. This validator is for such uses.

The given value is valid if it matches the regular expression specified in PATTERN_VALIDCHARACTERS.

Note: A value of NULL or an empty string (‘’) is considered valid

---

LocaleIdentifierValidator

A validator for locale identifiers.

This validator validates a string based on the expressions of the Flow I18n implementation.

Is valid if the given value is a valid “locale identifier”.

Note: A value of NULL or an empty string (‘’) is considered valid

---

MediaTypeValidator

The given $value is matches the defined media types Note: a value of NULL or empty string (‘’) are considered valid

The given $value is valid media type matches one of the allowedTypes and none of the disallowedTypes

Note: a value of NULL or empty string (‘’) is considered valid and was handled already

---

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `allowedTypes` (array): Array of allowed media ranges
- `disallowedTypes` (array): Array of disallowed media ranges

NotEmptyValidator

Validator for not empty values.

Checks if the given value is not empty (NULL, empty string, empty array or empty object that implements the Countable interface).

NumberRangeValidator

Validator for general numbers

The given value is valid if it is a number in the specified range.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `minimum` (integer, *optional*): The minimum value to accept
- `maximum` (integer, *optional*): The maximum value to accept

NumberValidator

Validator for general numbers.

Checks if the given value is a valid number.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `locale` (string|Locale, *optional*): The locale to use for number parsing
- `strictMode` (boolean, *optional*): Use strict mode for number parsing
- `formatLength` (string, *optional*): The format length, see NumbersReader::FORMAT_LENGTH_*
- `formatType` (string, *optional*): The format type, see NumbersReader::FORMAT_TYPE_*

RawValidator

A validator which accepts any input.

This validator is always valid.

Note: A value of NULL or an empty string (‘’) is considered valid

RegularExpressionValidator

Validator based on regular expressions.

Checks if the given value matches the specified regular expression.

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `regularExpression` (string): The regular expression to use for validation, used as given

StringLengthValidator

Validator for string length.

Checks if the given value is a valid string (or can be cast to a string if an object is given) and its length is between minimum and maximum specified in the validation options.

Note: A value of NULL or an empty string (‘’) is considered valid

---

Arguments

- `minimum` (integer, *optional*): Minimum length for a valid string
- `maximum` (integer, *optional*): Maximum length for a valid string
- `ignoreHtml` (boolean, *optional*): If true, HTML tags will be stripped before counting the characters

StringValidator

Validator for strings.

Checks if the given value is a string.

Note: A value of NULL or an empty string (‘’) is considered valid

---

TextValidator

Validator for “plain” text.

Checks if the given value is a valid text (contains no XML tags).

Be aware that the value of this check entirely depends on the output context. The validated text is not expected to be secure in every circumstance, if you want to be sure of that, use a customized regular expression or filter on output.

See http://php.net/filter_var for details.

Note: A value of NULL or an empty string (‘’) is considered valid

UniqueEntityValidator

Validator for uniqueness of entities.

Checks if the given value is a unique entity depending on it’s identity properties or custom configured identity properties.

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `identityProperties` (array, *optional*): List of custom identity properties.

UuidValidator

Validator for Universally Unique Identifiers.

Checks if the given value is a syntactically valid UUID.

Note: A value of NULL or an empty string (‘’) is considered valid

1.9.2 Media Validator Reference

This reference was automatically generated from code on 2024-03-10

ImageOrientationValidator

Validator that checks the orientation (square, portrait, landscape) of a given image.

Supported validator options are (array)allowedOrientations with one or two out of ‘square’, ‘landcape’ or ‘portrait’.

Example:

```
[at]Flow\Validate("$image", type="\Neos\Media\Validator\ImageOrientationValidator",
      options={ "allowedOrientations"={"square", "landscape"} })
```

this would refuse an image that is in portrait orientation, but allow landscape and square ones.

The given $value is valid if it is an NeosMediaDomainModelImageInterface of the configured orientation (square, portrait and/or landscape) Note: a value of NULL or empty string (‘’) is considered valid

Note: A value of NULL or an empty string (‘’) is considered valid

Arguments

- `allowedOrientations` (array): Array of image orientations, one or two out of 'square', 'landcape' or 'portrait'

ImageSizeValidator

Validator that checks size (resolution) of a given image

Example: [at]FlowValidate("$image", type="NeosMediaValidatorImageSizeValidator", options={ "minimumWidth"=150, "maximumResolution"=60000 })

The given $value is valid if it is an NeosMediaDomainModelImageInterface of the configured resolution Note: a value of NULL or empty string ('') is considered valid

Note: A value of NULL or an empty string ('') is considered valid

Arguments

- `minimumWidth` (integer, *optional*): The minimum width of the image
- `minimumHeight` (integer, *optional*): The minimum height of the image
- `maximumWidth` (integer, *optional*): The maximum width of the image
- `maximumHeight` (integer, *optional*): The maximum height of the image
- `minimumResolution` (integer, *optional*): The minimum resolution of the image
- `maximumResolution` (integer, *optional*): The maximum resolution of the image

ImageTypeValidator

Validator that checks the type of a given image

Example: [at]FlowValidate("$image", type="NeosMediaValidatorImageTypeValidator", options={ "allowedTypes"={"jpeg", "png"} })

The given $value is valid if it is an NeosMediaDomainModelImageInterface of the configured type (one of the image/* IANA media subtypes)

Note: a value of NULL or empty string ('') is considered valid

Note: A value of NULL or an empty string ('') is considered valid

Arguments

- `allowedTypes` (array): Allowed image types (using image/* IANA media subtypes)

1.9.3 Party Validator Reference

This reference was automatically generated from code on 2024-03-10

AimAddressValidator

Validator for AIM addresses.

Checks if the given value is a valid AIM name.

The AIM name has the following requirements: "It must be between 3 and 16 alphanumeric characters in length and must begin with a letter."

Note: A value of NULL or an empty string ('') is considered valid

---

IcqAddressValidator

Validator for ICQ addresses.

Checks if the given value is a valid ICQ UIN address.

The ICQ UIN address has the following requirements: "It must be 9 numeric characters." More information is found on: http://www.icq.com/support/icq_8/start/authorization/en

---

Note: A value of NULL or an empty string ('') is considered valid

---

JabberAddressValidator

Validator for Jabber addresses.

Checks if the given value is a valid Jabber name.

The Jabber address has the following structure: "name@jabber.org" More information is found on: http://tracker.phpbb.com/browse/PHPBB3-3832

Note: A value of NULL or an empty string ('') is considered valid

---

MsnAddressValidator

Validator for MSN addresses.

Checks if the given value is a valid MSN address.

The MSN address has the following structure: "name@hotmail.com, name@live.com, name@msn.com, name@outlook.com"

Note: A value of NULL or an empty string ('') is considered valid

SipAddressValidator

Validator for Sip addresses.

Checks if the given value is a valid Sip name.

The Sip address has the following structure: "sip:+4930432343@isp.com" More information is found on: http://wiki.snom.com/Features/Dial_Plan/Regular_Expressions

Note: A value of NULL or an empty string ('') is considered valid

SkypeAddressValidator

Validator for Skype addresses.

Checks if the given value is a valid Skype name.

The Skype website says: "It must be between 6-32 characters, start with a letter and contain only letters and numbers (no spaces or special characters)."

Nevertheless dash and underscore are allowed as special characters. Furthermore, account names can contain a colon if they were auto-created trough a connected Microsoft or Facebook profile. In this case, the syntax is as follows: - live:john.due - Facebook:john.doe

We added period and minus as additional characters because they are suggested by Skype during registration.

Note: A value of NULL or an empty string ('') is considered valid

UrlAddressValidator

Validator for URL addresses.

Checks if the given value is a valid URL.

Note: A value of NULL or an empty string ('') is considered valid

YahooAddressValidator

Validator for Yahoo addresses.

Checks if the given value is a valid Yahoo address.

The Yahoo address has the following structure: "name@yahoo.*"

Note: A value of NULL or an empty string ('') is considered valid

---

1.10 Signal Reference

1.10.1 Content Repository Signals Reference

This reference was automatically generated from code on 2024-03-10

Context (`Neos\ContentRepository\Domain\Service\Context`)

This class contains the following signals.

beforeAdoptNode

afterAdoptNode

Node (`Neos\ContentRepository\Domain\Model\Node`)

This class contains the following signals.

beforeNodeMove

afterNodeMove

beforeNodeCopy

afterNodeCopy

nodePathChanged

Autogenerated Proxy Method

Signals that the node path has been changed.

beforeNodeCreate

Autogenerated Proxy Method

Signals that a node will be created.

afterNodeCreate

Autogenerated Proxy Method

Signals that a node was created.

nodeAdded

Autogenerated Proxy Method

Signals that a node was added.

nodeUpdated

Autogenerated Proxy Method

Signals that a node was updated.

nodeRemoved

Autogenerated Proxy Method

Signals that a node was removed.

beforeNodePropertyChange

Autogenerated Proxy Method

Signals that the property of a node will be changed.

nodePropertyChanged

Autogenerated Proxy Method

Signals that the property of a node was changed.

NodeData (Neos\ContentRepository\Domain\Model\NodeData)

This class contains the following signals.

nodePathChanged

Autogenerated Proxy Method

Signals that a node has changed its path.

NodeDataRepository (Neos\ContentRepository\Domain\Repository\NodeDataRepository)

This class contains the following signals.

repositoryObjectsPersisted

Autogenerated Proxy Method

Signals that persistEntities() in this repository finished correctly.

PaginateController (Neos\ContentRepository\ViewHelpers\Widget\Controller\PaginateController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

PublishingService (Neos\ContentRepository\Domain\Service\PublishingService)

This class contains the following signals.

nodePublished

Autogenerated Proxy Method

Signals that a node has been published.

The signal emits the source node and target workspace, i.e. the node contains its source workspace.

nodeDiscarded

Autogenerated Proxy Method

Signals that a node has been discarded.

The signal emits the node that has been discarded.

Workspace (`Neos\ContentRepository\Domain\Model\Workspace`)

This class contains the following signals.

baseWorkspaceChanged

Autogenerated Proxy Method

Emits a signal after the base workspace has been changed

beforeNodePublishing

Autogenerated Proxy Method

Emits a signal just before a node is being published

The signal emits the source node and target workspace, i.e. the node contains its source workspace.

afterNodePublishing

Autogenerated Proxy Method

Emits a signal when a node has been published.

The signal emits the source node and target workspace, i.e. the node contains its source workspace.

1.10.2 Flow Signals Reference

This reference was automatically generated from code on 2024-03-10

AbstractAdvice (`Neos\Flow\Aop\Advice\AbstractAdvice`)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

AbstractAuthenticationController (Neos\Flow\Security\Authentication\Controller\AbstractAuthenticationController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ActionController (Neos\Flow\Mvc\Controller\ActionController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ActionRequest (Neos\Flow\Mvc\ActionRequest)

This class contains the following signals.

requestDispatched

Autogenerated Proxy Method

Emits a signal when a Request has been dispatched

The action request is not proxyable, so the signal is dispatched manually here. The safeguard allows unit tests without the dispatcher dependency.

AfterAdvice (Neos\Flow\Aop\Advice\AfterAdvice)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

AfterReturningAdvice (Neos\Flow\Aop\Advice\AfterReturningAdvice)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

AfterThrowingAdvice (Neos\Flow\Aop\Advice\AfterThrowingAdvice)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

AroundAdvice (Neos\Flow\Aop\Advice\AroundAdvice)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

AuthenticationProviderManager (Neos\Flow\Security\Authentication\AuthenticationProviderManager)

This class contains the following signals.

authenticatedToken

Autogenerated Proxy Method

Signals that the specified token has been successfully authenticated.

loggedOut

Autogenerated Proxy Method

Signals that all active authentication tokens have been invalidated. Note: the session will be destroyed after this signal has been emitted.

successfullyAuthenticated

Autogenerated Proxy Method

Signals that authentication commenced and at least one token was authenticated.

BeforeAdvice (Neos\Flow\Aop\Advice\BeforeAdvice)

This class contains the following signals.

adviceInvoked

Emits a signal when an Advice is invoked

The advice is not proxyable, so the signal is dispatched manually here.

Bootstrap (Neos\Flow\Core\Bootstrap)

This class contains the following signals.

finishedCompiletimeRun

Emits a signal that the compile run was finished.

finishedRuntimeRun

Emits a signal that the runtime run was finished.

bootstrapShuttingDown

Emits a signal that the bootstrap finished and is shutting down.

CacheCommandController (Neos\Flow\Command\CacheCommandController)

This class contains the following signals.

warmupCaches

Autogenerated Proxy Method

Signals that caches should be warmed up.

Other application parts may subscribe to this signal and execute additional tasks for preparing the application for the first request.

Compiler (Neos\Flow\ObjectManagement\Proxy\Compiler)

This class contains the following signals.

compiledClasses

ConfigurationManager (Neos\Flow\Configuration\ConfigurationManager)

This class contains the following signals.

configurationManagerReady

Emits a signal after The ConfigurationManager has been loaded

CoreCommandController (Neos\Flow\Command\CoreCommandController)

This class contains the following signals.

finishedCompilationRun

Signals that the compile command was successfully finished.

Dispatcher (Neos\Flow\Mvc\Dispatcher)

This class contains the following signals.

beforeControllerInvocation

Autogenerated Proxy Method

This signal is emitted directly before the request is been dispatched to a controller.

afterControllerInvocation

Autogenerated Proxy Method

This signal is emitted directly after the request has been dispatched to a controller and the controller returned control back to the dispatcher.

DoctrineCommandController (Neos\Flow\Command\DoctrineCommandController)

This class contains the following signals.

afterDatabaseMigration

EntityManagerFactory (Neos\Flow\Persistence\Doctrine\EntityManagerFactory)

This class contains the following signals.

beforeDoctrineEntityManagerCreation

afterDoctrineEntityManagerCreation

PackageManager (`Neos\Flow\Package\PackageManager`)

This class contains the following signals.

packageStatesUpdated

Emits a signal when package states have been changed (e.g. when a package was created)

The advice is not proxyable, so the signal is dispatched manually here.

PersistenceManager (`Neos\Flow\Persistence\Doctrine\PersistenceManager`)

This class contains the following signals.

allObjectsPersisted

Autogenerated Proxy Method

Signals that all persistAll() has been executed successfully.

PolicyService (`Neos\Flow\Security\Policy\PolicyService`)

This class contains the following signals.

configurationLoaded

Autogenerated Proxy Method

Emits a signal when the policy configuration has been loaded

This signal can be used to add roles and/or privilegeTargets during runtime. In the slot make sure to receive the $policyConfiguration array by reference so you can alter it.

rolesInitialized

Autogenerated Proxy Method

Emits a signal when roles have been initialized

This signal can be used to register roles during runtime. In the slot make sure to receive the $roles array by reference so you can alter it.

RestController (`Neos\Flow\Mvc\Controller\RestController`)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

SlaveRequestHandler (`Neos\Flow\Cli\SlaveRequestHandler`) (deprecated)

This class contains the following signals.

dispatchedCommandLineSlaveRequest

Emits a signal that a CLI slave request was dispatched.

DEPRECATED This will probably move to a separate package and be renamed in a future version, you should not rely on it.

StandardController (`Neos\Flow\Mvc\Controller\StandardController`)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

1.10.3 Media Signals Reference

This reference was automatically generated from code on 2024-03-10

AssetCollectionController (`Neos\Media\Browser\Controller\AssetCollectionController`)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AssetController (Neos\Media\Browser\Controller\AssetController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AssetProxyController (Neos\Media\Browser\Controller\AssetProxyController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AssetService (Neos\Media\Domain\Service\AssetService)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

Signals that an asset was added.

assetRemoved

Autogenerated Proxy Method

Signals that an asset was removed.

assetUpdated

Autogenerated Proxy Method

Signals that an asset was updated.

assetResourceReplaced

Autogenerated Proxy Method

Signals that a resource on an asset has been replaced

Note: when an asset resource is replaced, the assetUpdated signal is sent anyway and can be used instead.

ImageController (Neos\Media\Browser\Controller\ImageController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ImageVariantController (Neos\Media\Browser\Controller\ImageVariantController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

PaginateController (Neos\Media\Browser\ViewHelpers\Controller\PaginateController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

TagController (Neos\Media\Browser\Controller\TagController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ThumbnailController (Neos\Media\Controller\ThumbnailController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ThumbnailService (Neos\Media\Domain\Service\ThumbnailService)

This class contains the following signals.

thumbnailRefreshed

Autogenerated Proxy Method

Signals that a thumbnail was refreshed.

thumbnailPersisted

Autogenerated Proxy Method

Signals that a thumbnail was persisted.

thumbnailCreated

Autogenerated Proxy Method

Signals that a thumbnail was created.

UsageController (Neos\Media\Browser\Controller\UsageController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

1.10.4 Neos Signals Reference

This reference was automatically generated from code on 2024-03-10

AbstractCreate (Neos\Neos\Ui\Domain\Model\Changes\AbstractCreate)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

Signals, that all changes by node creation handlers are applied

AbstractModuleController (Neos\Neos\Controller\Module\AbstractModuleController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AbstractServiceController (Neos\Neos\Service\Controller\AbstractServiceController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AdministrationController (Neos\Neos\Controller\Module\AdministrationController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AssetProxiesController (Neos\Neos\Controller\Service\AssetProxiesController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

AssetsController (Neos\Neos\Controller\Service\AssetsController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

BackendController (Neos\Neos\Controller\Backend\BackendController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

BackendController (Neos\Neos\Ui\Controller\BackendController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

BackendServiceController (Neos\Neos\Ui\Controller\BackendServiceController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ConfigurationController (Neos\Neos\Controller\Module\Administration\ConfigurationController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ContentContext (Neos\Neos\Domain\Service\ContentContext)

This class contains the following signals.

beforeAdoptNode

afterAdoptNode

ContentController (Neos\Neos\Controller\Backend\ContentController)

This class contains the following signals.

assetUploaded

Autogenerated Proxy Method

Signals that a new asset has been uploaded through the Neos Backend

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ContentDimensionsController (Neos\Neos\Controller\Service\ContentDimensionsController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

Create (Neos\Neos\Ui\Domain\Model\Changes\Create)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

Signals, that all changes by node creation handlers are applied

CreateAfter (Neos\Neos\Ui\Domain\Model\Changes\CreateAfter)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

Signals, that all changes by node creation handlers are applied

CreateBefore (Neos\Neos\Ui\Domain\Model\Changes\CreateBefore)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

Signals, that all changes by node creation handlers are applied

DataSourceController (Neos\Neos\Service\Controller\DataSourceController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

DimensionController (Neos\Neos\Controller\Module\Administration\DimensionController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

HistoryController (Neos\Neos\Controller\Module\Management\HistoryController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ImpersonateController (Neos\Neos\Controller\Backend\ImpersonateController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

LoginController (Neos\Neos\Controller\LoginController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ManagementController (Neos\Neos\Controller\Module\ManagementController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

ModuleController (Neos\Neos\Controller\Backend\ModuleController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

NodeController (Neos\Neos\Controller\Frontend\NodeController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

NodeController (Neos\Neos\Service\Controller\NodeController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

NodesController (Neos\Neos\Controller\Service\NodesController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

PackagesController (Neos\Neos\Controller\Module\Administration\PackagesController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

PublishingService (Neos\Neos\Service\PublishingService)

This class contains the following signals.

nodePublished

Autogenerated Proxy Method

Signals that a node has been published.

The signal emits the source node and target workspace, i.e. the node contains its source workspace.

nodeDiscarded

Autogenerated Proxy Method

Signals that a node has been discarded.

The signal emits the node that has been discarded.

SchemaController (Neos\Neos\Controller\Backend\SchemaController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

SettingsController (Neos\Neos\Controller\Backend\SettingsController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

Site (Neos\Neos\Domain\Model\Site)

This class contains the following signals.

siteChanged

Autogenerated Proxy Method

Internal signal

SiteImportService (Neos\Neos\Domain\Service\SiteImportService)

This class contains the following signals.

siteImported

Autogenerated Proxy Method

Signal that is triggered when a site has been imported successfully

SiteService (Neos\Neos\Domain\Service\SiteService)

This class contains the following signals.

sitePruned

Autogenerated Proxy Method

Signal that is triggered whenever a site has been pruned

SitesController (Neos\Neos\Controller\Module\Administration\SitesController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

UserController (Neos\Neos\Controller\Module\UserController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

UserPreferenceController (Neos\Neos\Service\Controller\UserPreferenceController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

UserService (Neos\Neos\Domain\Service\UserService)

This class contains the following signals.

userCreated

Autogenerated Proxy Method

Signals that a new user, including a new account has been created.

userDeleted

Autogenerated Proxy Method

Signals that the given user has been deleted.

userUpdated

Autogenerated Proxy Method

Signals that the given user data has been updated.

rolesAdded

Autogenerated Proxy Method

Signals that new roles have been assigned to the given account

rolesRemoved

Autogenerated Proxy Method

Signals that roles have been removed to the given account

userActivated

Autogenerated Proxy Method

Signals that the given user has been activated

userDeactivated

Autogenerated Proxy Method

Signals that the given user has been activated

UserSettingsController (Neos\Neos\Controller\Module\User\UserSettingsController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

UsersController (Neos\Neos\Controller\Module\Administration\UsersController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

WorkspaceController (Neos\Neos\Service\Controller\WorkspaceController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

WorkspacesController (Neos\Neos\Controller\Module\Management\WorkspacesController)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

WorkspacesController (`Neos\Neos\Controller\Service\WorkspacesController`)

This class contains the following signals.

viewResolved

Autogenerated Proxy Method

Emit that the view is resolved. The passed ViewInterface reference, gives the possibility to add variables to the view, before passing it on to further rendering

1.11 Coding Guideline Reference

1.11.1 PHP Coding Guidelines & Best Practices

Coding Standards are an important factor for achieving a high code quality. A common visual style, naming conventions and other technical settings allow us to produce a homogenous code which is easy to read and maintain. However, not all important factors can be covered by rules and coding standards. Equally important is the style in which certain problems are solved programmatically - it's the personality and experience of the individual developer which shines through and ultimately makes the difference between technically okay code or a well considered, mature solution.

These guidelines try to cover both, the technical standards as well as giving incentives for a common development style. These guidelines must be followed by everyone who creates code for the Flow core. Because Neos is based on Flow, it follows the same principles - therefore, whenever we mention Flow in the following sections, we equally refer to Neos. We hope that you feel encouraged to follow these guidelines as well when creating your own packages and Flow based applications.

CGL on One Page

Fig. 1: The Coding Guidelines on One Page

The most important parts of our `Coding Guidelines in a one page document` you can print out and hang on your wall for easy reference. Does it get any easier than that?

Code Formatting and Layout aka "beautiful code"

The visual style of programming code is very important. In the Neos project we want many programmers to contribute, but in the same style. This will help us to:

- Easily read/understand each others code and consequently easily spot security problems or optimization opportunities
- It is a signal about consistency and cleanliness, which is a motivating factor for programmers striving for excellence

Some people may object to the visual guidelines since everyone has his own habits. You will have to overcome that in the case of Flow; the visual guidelines must be followed along with coding guidelines for security. We want all contributions to the project to be as similar in style and as secure as possible.

General considerations

- Follow the PSR-2 standard for code formatting
- Almost every PHP file in Flow contains exactly one class and does not output anything if it is called directly. Therefore you start your file with a `<?php` tag and must not end it with the closing `?>`.
- Every file must contain a header stating namespace and licensing information
  - Declare your namespace.
  - The copyright header itself must not start with `/**`, as this may confuse documentation generators!

The Flow standard file header:

```
<?php
namespace YourCompany\Package\Something\New;

/*
 * This file is part of the YourCompany.Package package.
 *
 * (c) YourCompany
 *
 * This package is Open Source Software. For the full copyright and license
 * information, please view the LICENSE file which was distributed with this
 * source code.
 */
```

- Code lines are of arbitrary length, no strict limitations to 80 characters or something similar (wake up, graphical displays have been available for decades now...). But feel free to break lines for better readability if you think it makes sense!
- Lines end with a newline a.k.a `chr(10)` - UNIX style
- Files must be encoded in UTF-8 without byte order mark (BOM)

Make sure you use the correct license and mention the correct package in the header.

Indentation and line formatting

Since we adopted PSR-2 as coding standard we use spaces for indentation.

Here's a code snippet which shows the correct usage of spaces.

Correct use of indentation:

```
/**
 * Returns the name of the currently set context.
 *
 * @return string Name of the current context
 */
public function getContextName()
{
    return $this->contextName;
}
```

Naming

Naming is a repeatedly undervalued factor in the art of software development. Although everybody seems to agree on that nice names are a nice thing to have, most developers choose cryptic abbreviations in the end (to save some typing). Beware that we Neos core developers are very passionate about naming (some people call it fanatic, well ...). In our opinion spending 15 minutes (or more ...) just to find a good name for a method is well spent time! There are zillions of reasons for using proper names and in the end they all lead to better readable, manageable, stable and secure code.

As a general note, english words (or abbreviations if necessary) must be used for all class names, method names, comments, variables names, database table and field names. The consensus is that english is much better to read for the most of us, compared to other languages.

When using abbreviations or acronyms remember to make them camel-cased as needed, no all-uppercase stuff.

Vendor namespaces

The base for namespaces as well as package keys is the vendor namespace. Since Flow is part of the Neos project, the core team decided to choose "Neos" as our vendor namespace. The Object Manager for example is known under the class name `Neos\Flow\ObjectManagement\ObjectManager`. In our examples you will find the `Acme` vendor namespace.

Why do we use vendor namespaces? This has two great benefits: first of all we don't need a central package key registry and secondly, it allows anyone to seamlessly integrate third-party packages, such as Symfony2 components and Zend Framework components or virtually any other PHP library.

Think about your own vendor namespace for a few minutes. It will stay with you for a long time.

Package names

All package names start with an uppercase character and usually are written in `UpperCamelCase`. In order to avoid problems with different filesystems, only the characters a-z, A-Z, 0-9 and the dash sign "-" are allowed for package names – don't use special characters.

The full package key is then built by combining the vendor namespace and the package, like `Neos.Eel` or `Acme.Demo`.

Namespace and Class names

- Only the characters a-z, A-Z and 0-9 are allowed for namespace and class names.
- Namespaces are usually written in UpperCamelCase but variations are allowed for well established names and abbreviations.
- Class names are always written in `UpperCamelCase`.
- The unqualified class name must be meant literally even without the namespace.
- The main purpose of namespaces is categorization and ordering
- Class names must be nouns, never adjectives.
- The name of abstract classes must start with the word "Abstract", class names of aspects must end with the word "Aspect".

Incorrect naming of namespaces and classes

Fully qualified class name	Unqualified name	Remarks
\Neos\Flow\Session\Php	Php	The class is not a representation of PHP
\Neos\Cache\Backend\File	File	The class doesn't represent a file!
\Neos\Flow\Session\Interface	Interface	Not allowed, "Interface" is a reserved keyword
\Neos\Foo\Controller\Default	Default	Not allowed, "Default" is a reserved keyword
\Neos\Flow\Objects\Manager	Manager	Just "Manager" is too fuzzy

Correct naming of namespaces and classes

Fully qualified class name	Unqualified name	Remarks
\Neos\Flow\Session\PhpSession	PhpSession	That's a PHP Session
\Neos\Flow\Cache\Backend\FileBackend	FileBackend	A File Backend
\Neos\Flow\Session\SessionInterface	SessionInterface	Interface for a session
\Neos\Foo\Controller\StandardController	StandardController	The standard controller
\Neos\Flow\Objects\ObjectManager	ObjectManager	"ObjectManager" is clearer

Edge cases in naming of namespaces and classes

Fully qualified class name	Unqualified name	Remarks
\Neos\Flow\Mvc\ControllerInterfa	ControllerInterface	Consequently the interface belongs to all the controllers in the Controller sub namespace
\Neos\Flow\Mvc\Controller\Contr		Better
\Neos\Cache\AbstractBackend	AbstractBackend	Same here: In reality this class belongs to the backends
\Neos\Cache\Backend\AbstractBac		Better

Note: When specifying class names to PHP, always reference the global namespace inside namespaced code by using a leading backslash. When referencing a class name inside a string (e.g. given to the `get`-Method of the `ObjectManager`, in pointcut expressions or in YAML files), never use a leading backslash. This follows the native PHP notion of names in strings always being seen as fully qualified.

Importing Namespaces

If you refer to other classes or interfaces you are encouraged to import the namespace with the `use` statement if it improves readability.

Following rules apply:

- If importing namespaces creates conflicting class names you might alias class/interface or namespaces with the `as` keyword.
- One `use` statement per line, one `use` statement for each imported namespace
- Imported namespaces should be ordered alphabetically (modern IDEs provide support for this)

Tip: `use` statements have no side-effects (e.g. they don't trigger autoloading). Nevertheless you should remove unused imports for better readability

Interface names

Only the characters a-z, A-Z and 0-9 are allowed for interface names – don't use special characters.

All interface names are written in `UpperCamelCase`. Interface names must be adjectives or nouns and have the Interface suffix. A few examples follow:

- `\Neos\Flow\ObjectManagement\ObjectInterface`
- `\Neos\Flow\ObjectManagement\ObjectManagerInterface`
- `\MyCompany\MyPackage\MyObject\MySubObjectInterface`
- `\MyCompany\MyPackage\MyObject\MyHtmlParserInterface`

Exception names

Exception naming basically follows the rules for naming classes. There are two possible types of exceptions: generic exceptions and specific exceptions. Generic exceptions should be named "Exception" preceded by their namespace. Specific exceptions should reside in their own sub-namespace end with the word `Exception`.

- `\Neos\Flow\ObjectManagement\Exception`
- `\Neos\Flow\ObjectManagement\Exception\InvalidClassNameException`
- `\MyCompany\MyPackage\MyObject\Exception`
- `\MyCompany\MyPackage\MyObject\Exception\OutOfCoffeeException`

On consistent naming of classes, interfaces and friends

At times, the question comes up, why we use a naming scheme that is inconsistent with what we write in the PHP sources. Here is the best explanation we have:

> At first glance this feels oddly inconsistent; We do, after all, put each of those at the same position within php code.
>
> But, I think leaving Abstract as a prefix, and Interface/Trait as suffixes makes sense. Consider the opposite of how we do it: "Interface Foo", "Trait Foo" both feel slightly odd when I say them out loud, and "Foo Abstract" feels very wrong. I think that is because of the odd rules of grammar in English (Oh! English. What an ugly inconsistent language! And yet, it is my native tongue).
>
> Consider the phrase "the poor man". 'poor' is an adjective that describes 'man', a noun. Poor happens to also work as a noun, but the definition changes slightly when you use it as a noun instead of an adjective. And, if you were to flip the phrase around, it would not make much sense, or could have (sometimes funny) alternative meanings: "the man poor" (Would that mean someone without a boyfriend?)
>
> The word "Abstract" works quite well as an adjective, but has the wrong meaning as a noun. An "Abstract" (noun) is "an abridgement or summary" or a kind of legal document, or any other summary-like document. But we're not talking about a document, we're talking about the computing definition which is an adjective: "abstract type". (http://en.wiktionary.org/wiki/abstract)
>
> "Abstract" can be a noun, an adjective, or a verb. But, we want the adjective form. "Interface" is a noun or a verb. "Trait" is always a noun. So, based on current English rules, "Abstract Foo", "Foo Interface" and "Foo Trait" feel the most natural. English is a living language where words can move from one part of speech to another, so we could get away with using the words in different places in the sentence. But that would, at least to begin with, feel awkward.
>
> So, I blame the inconsistent placement of Abstract, Interface, and Trait on the English language.
>
> [...]
>
> —Jacob Floyd, http://lists.typo3.org/pipermail/flow/2014-November/005625.html

Method names

All method names are written in lowerCamelCase. In order to avoid problems with different filesystems, only the characters a-z, A-Z and 0-9 are allowed for method names – don't use special characters.

Make method names descriptive, but keep them concise at the same time. Constructors must always be called `__construct()`, never use the class name as a method name.

- `myMethod()`
- `someNiceMethodName()`
- `betterWriteLongMethodNamesThanNamesNobodyUnderstands()`
- `singYmcaLoudly()`
- `__construct()`

Variable names

Variable names are written in `lowerCamelCase` and should be

- self-explanatory
- not shortened beyond recognition, but rather longer if it makes their meaning clearer

The following example shows two variables with the same meaning but different naming. You'll surely agree the longer versions are better (don't you . . . ?).

Correct naming of variables

- `$singletonObjectsRegistry`
- `$argumentsArray`
- `$aLotOfHtmlCode`

Incorrect naming of variables

- `$sObjRgstry`
- `$argArr`
- `$cx`

As a special exception you may use variable names like `$i`, `$j` and `$k` for numeric indexes in `for` loops if it's clear what they mean on the first sight. But even then you should want to avoid them.

Constant names

All constant names are written in `UPPERCASE`. This includes `TRUE`, `FALSE` and `NULL`. Words can be separated by underscores - you can also use the underscore to group constants thematically:

- `STUFF_LEVEL`
- `COOLNESS_FACTOR`
- `PATTERN_MATCH_EMAILADDRESS`
- `PATTERN_MATCH_VALIDHTMLTAGS`

It is, by the way, a good idea to use constants for defining regular expression patterns (as seen above) instead of defining them somewhere in your code.

Filenames

These are the rules for naming files:

- All filenames are `UpperCamelCase`.
- Class and interface files are named according to the class or interface they represent
- Each file must contain only one class or interface
- Names of files containing code for unit tests must be the same as the class which is tested, appended with "Test.php".
- Files are placed in a directory structure representing the namespace structure. You may use PSR-0 or PSR-4 autoloading as you like. We generally use PSR-4.

File naming in Flow

`Neos.TemplateEngine/Classes/TemplateEngineInterface.php`
Contains the interface `\Neos\TemplateEngine\TemplateEngineInterface` which is part of the package *Neos.TemplateEngine*

`Neos.Flow/Classes/Error/RuntimeException.php`
Contains the `\Neos\Flow\Error\Messages\RuntimeException` being a part of the package *Neos.Flow*

`Acme.DataAccess/Classes/CustomQuery.php`
Contains class `\Acme\DataAccess\CustomQuery` which is part of the package *Acme.DataAccess*

`Neos.Flow/Tests/Unit/Package/PackageManagerTest.php`
Contains the class `\Neos\Flow\Tests\Unit\Package\PackageManagerTest` which is a PHPUnit testcase for `Package\PackageManager`.

PHP code formatting

PSR-2

We follow the PSR-2 standard which is defined by PHP FIG. You should read the full *PSR-2 standard*. .. psr-2 standard: https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

Some things are not specified in PSR-2, so here are some amendments.

Strings

In general, we use single quotes to enclose literal strings:

```
$neos = 'A great project from a great team';
```

If you'd like to insert values from variables, concatenate strings. A space must be inserted before and after the dot for better readability:

```
$message = 'Hey ' . $name . ', you look ' . $appearance . ' today!';
```

You may break a string into multiple lines if you use the dot operator. You'll have to indent each following line to mark them as part of the value assignment:

```
$neos = 'A great ' .
  'project from ' .
  'a great ' .
  'team';
```

You should also consider using a PHP function such as *sprintf()* to concatenate strings to increase readability:

```
$message = sprintf('Hey %s, you look %s today!', $name, $appearance);
```

Development Process

Test-Driven Development

In a nutshell: before coding a feature or fixing a bug, write an unit test.

Whatever you do: before committing changes to the repository, run all unit tests to make sure nothing is broken!

Commit Messages

To have a clear and focused history of code changes is greatly helped by using a consistent way of writing commit messages. Because of this and to help with (partly) automated generation of change logs for each release we have defined a fixed syntax for commit messages that is to be used.

Tip: Never commit without a commit message explaining the commit!

The syntax is as follows:

- Start with one of the following codes:

  FEATURE:
  A feature change. Most likely it will be an added feature, but it could also be removed. For additions there should be a corresponding ticket in the issue tracker.

  BUGFIX:
  A fix for a bug. There should be a ticket corresponding to this in the issue tracker as well as a new) unit test for the fix.

  SECURITY:
  A security related change. Those must only be committed by active contributors in agreement with the Neos Security Team.

  TASK:
  Anything not covered by the above categories, e.g. coding style cleanup or documentation changes. Usually only used if there's no corresponding ticket.

  Except for SECURITY each of the above codes can be prefixed with WIP to mark a change **work in progress**. This means that it is not yet ready for a final review. The WIP prefix must be removed before a change is merged.

- The code is followed by a short summary in the same line, no full stop at the end. If the change affects the public API or is likely to break things on the user side, start the line with **[!!!]**. This indicates a breaking change that needs human action when updating. Make sure to explain why a change is breaking and in what circumstances.
- Then follows (after a blank line) a custom message explaining what was done. It should be written in a style that serves well for a change log read by users.

- If there is more to say about a change add a new paragraph with background information below. In case of breaking changes give a hint on what needs to be changed by the user.
- If corresponding tickets exist, mention the ticket number(s) using footer lines after another blank line and use the following actions:

  Fixes <Issue-Id>
  If the change fixes a bug, resolves a feature request or task.

  Related to <Issue-Id>
  If the change relates to an issue but does not resolve or fix it.

A commit messages following the rules...:

```
TASK: Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the present tense: "Fix bug" and not "Fixed
bug."  This convention matches up with commit messages generated by
commands like git merge and git revert.

Code snippets::

 should be written in
 ReStructuredText compatible
 format for better highlighting

Further paragraphs come after blank lines.

* Bullet points are okay, too
* An asterisk is used for the bullet, it can be preceded by a single
  space. This format is rendered correctly by Forge (redmine)
* Use a hanging indent

Fixes #123
```

Examples of good and bad subject lines:

```
Introduce xyz service                                // BAD, missing code prefix
BUGFIX: Fixed bug xyz                                // BAD, subject should be written in␣
↪present tense
WIP !!! TASK: A breaking change                      // BAD, subject has to start with [!!
↪!] for breaking changes
BUGFIX: Make SessionManager remove expired sessions // GOOD, the line explains what the␣
↪change does, not what the
                                                        bug is about (this should be␣
↪explained in the following lines
                                                        and in the related bug tracker␣
↪ticket)
```

Source Code Documentation

All code must be documented with inline comments. The syntax is similar to that known from the Java programming language (JavaDoc). This way code documentation can automatically be generated.

Documentation Blocks

A file contains different documentation blocks, relating to the class in the file and the members of the class. A documentation block is always used for the entity it precedes.

Class documentation

Classes have their own documentation block describing the classes purpose.

Standard documentation block:

```
/**
 * First sentence is short description. Then you can write more, just as you like
 *
 * Here may follow some detailed description about what the class is for.
 *
 * Paragraphs are separated by an empty line.
 */
class SomeClass {
 ...
}
```

Additional tags or annotations, such as `@see` or `@Flow\Aspect`, can be added as needed.

Documenting variables, constants, includes

Properties of a class should be documented as well. We use the short version for documenting them.

Standard variable documentation block:

```
/**
 * A short description, very much recommended
 *
 * @var string
 */
protected $title = 'Untitled';
```

In general you should try to code in a way that the types can be derived (e.g. by using type hints and annotations). In some cases this is not possible, for example when iterating through an array of objects. In these cases it's ok to add inline @var annotations to increase readability and to activate auto-completion and syntax-highlighting:

```
protected function someMethod(array $products) {
   /** @var $product \Acme\SomePackage\Domain\Model\Product */
   foreach ($products as $product) {
       $product->getTitle();
   }
}
```

Method documentation

For a method, at least all parameters and the return value must be documented.

Standard method documentation block:

```
/**
 * A description for this method
 *
 * Paragraphs are separated by an empty line.
 *
 * @param \Neos\Blog\Domain\Model\Post $post A post
 * @param string $someString This parameter should contain some string
 * @return void
 */
public function addStringToPost(\Neos\Blog\Domain\Model\Post $post, $someString) {
 ...
}
```

A special note about the `@param` tags: The parameter type and name are separated by one space, not aligned. Do not put a colon after the parameter name. Always document the return type, even if it is void - that way it is clearly visible it hasn't just been forgotten (only constructors never have a `@return` annotation!).

Testcase documentation

Testcases need to be marked as being a test and can have some more annotations.

Standard testcase documentation block:

```
/**
 * @test
 */
public function fooReturnsBarForQuux() {
 ...
}
```

Defining the Public API

Not all methods with a public visibility are necessarily part of the intended public API of a project. For Flow, only the methods explicitly defined as part of the public API will be kept stable and are intended for use by developers using Flow. Also the API documentation we produce will only cover the public API.

To mark a method as part of the public API, include an `@api` annotation for it in the docblock.

Defining the public API:

```
/**
 * This method is part of the public API.
 *
 * @return void
 * @api
 */
public function fooBar() {
```

(continues on next page)

(continued from previous page)

```
  ...
}
```

Tip: When something in a class or an interface is annotated with `@api` make sure to also annotate the class or interface itself! Otherwise it will be ignored completely when official API documentation is rendered!

Overview of Documentation Annotations

There are not only documentation annotations that can be used. In Flow annotations are also used in the MVC component, for defining aspects and advices for the AOP framework as well as for giving instructions to the Persistence framework. See the individual chapters for information on their purpose and use.

Here is a list of annotations used within the project. They are grouped by use case and the order given here should be kept for the sake of consistency.

Interface Documentation

- @api
- @since
- @deprecated

Class Documentation

- @FlowIntroduce
- @FlowEntity
- @FlowValueObject
- @FlowScope
- @FlowAutowiring
- @FlowLazy
- @FlowAspect
- @api
- @since
- @deprecated

Property Documentation

- @FlowIntroduce
- @FlowIdentity
- @FlowTransient
- @FlowLazy
- @FlowIgnoreValidation
- @FlowInject
- @FlowInjectConfiguration
- @FlowValidate

- @var
- @api
- @since
- @deprecated

Constructor Documentation

- @param
- @throws
- @api
- @since
- @deprecated

Method Documentation

- @FlowAfter
- @FlowAfterReturning
- @FlowAfterThrowing
- @FlowAround
- @FlowBefore
- @FlowPointcut
- @FlowAutowiring
- @FlowCompileStatic
- @FlowFlushesCaches
- @FlowInternal
- @FlowSession
- @FlowSignal
- @FlowIgnoreValidation
- @FlowSkipCsrfProtection
- @FlowValidate
- @FlowValidationGroups
- @param
- @return
- @throws
- @api
- @since
- @deprecated

Testcase Documentation

- @test
- @dataProvider

- @expectedException

Tip: Additional annotations (more or less only the `@todo` and `@see` come to mind here), should be placed after all other annotations.

Best Practices

Flow

This section gives you an overview of Flow's coding rules and best practices.

Error Handling and Exceptions

Flow makes use of a hierarchy for its exception classes. The general rule is to throw preferably specific exceptions and usually let them bubble up until a place where more general exceptions are caught. Consider the following example:

Some method tried to retrieve an object from the object manager. However, instead of providing a string containing the object name, the method passed an object (of course not on purpose - something went wrong). The object manager now throws an `InvalidObjectName` exception. In order to catch this exception you can, of course, catch it specifically - or only consider a more general `Object` exception (or an even more general `Flow` exception). This all works because we have the following hierarchy:

```
+ \Neos\Flow\Exception
+ \Neos\Flow\ObjectManagement\Exception
+ \Neos\Flow\ObjectManagement\Exception\InvalidObjectNameException
```

Throwing an exception

When throwing an exception, make sure to provide a clear error message and an *error code being the unix timestamp of when you write the ``throw`` statement*. That error code must be unique, so watch out when doing copy and paste!

Unit Testing

Some notes for a start:

- Never use the object manager or factory in unit tests! If they are needed, mock them.
- Avoid tests for the scope of an object. Those tests test the object factory, rather then the test target. Such a test should be done by checking for the presence of an expected @scope annotation – eventually we will find an elegant way for this.

Cross Platform Coding

- When concatenating paths, always use `\Neos\Utility\Files::concatenatePaths()` to avoid trouble.

PHP in General

- All code should be object oriented. This means there should be no functions outside classes if not absolutely necessary. If you need a "container" for some helper methods, consider creating a static class.
- All code must make use of PHP5 advanced features for object oriented programming.
  - Use PHP namespaces
  - Always declare the scope (public, protected, private) of methods and member variables
  - Make use of iterators and exceptions, have a look at the SPL
- Make use of type-hinting wherever possible
- Always use `<?php` as opening tags (never only `<?`)
- Never use the closing tag `?>` at the end of a file, leave it out
- Never use the shut-up operator `@` to suppress error messages. It makes debugging harder, is dirty style and slow as hell
- Prefer strict comparisons whenever possible, to avoid problems with truthy and falsy values that might behave different than what you expect. Here are some examples:

  Examples of good and bad comparisons:

```
if ($template)              // BAD
if (isset($template))       // GOOD
if ($template !== NULL))    // GOOD
if ($template !== ''))      // GOOD

if (strlen($template) > 0) // BAD! strlen("-1") is greater than 0
if (is_string($template) && strlen($template) > 0) // BETTER

if ($foo == $bar)           // BAD, avoid truthy comparisons
if ($foo != $bar)           // BAD, avoid falsy comparisons
if ($foo === $bar))         // GOOD
if ($foo !== $bar))         // GOOD
```

- Order of methods in classes. To gain a better overview, it helps if methods in classes are always ordered in a certain way. We prefer the following:
  - constructor
  - injection methods
  - initialization methods (including `initializeObject()`)
  - public methods
  - protected methods
  - private methods
  - shutdown methods
  - destructor

Fig. 2: Truthy and falsy are fuzzy…

- Avoid double-negation. Instead of `exportSystemView(..., $noRecurse)` use `exportSystemView(..., $recurse)`. It is more logical to pass `TRUE` if you want recursion instead of having to pass `FALSE`. In general, parameters negating things are a bad idea.

Comments

In general, comments are a good thing and we strive for creating a well-documented source code. However, inline comments can often be a sign for a bad code structure or method naming.[1] As an example, consider the example for a coding smell:

```
 // We only allow valid persons
if (is_object($p) && strlen($p->lastN) > 0 && $p->hidden === FALSE && $this->environment-
↪>moonPhase === MOON_LIB::CRESCENT) {
 $xmM = $thd;
}
```

This is a perfect case for the refactoring technique "extract method": In order to avoid the comment, create a new method which is as explanatory as the comment:

```
if ($this->isValidPerson($person) {
  $xmM = $thd;
}
```

Bottom line is: You may (and are encouraged to) use inline comments if they support the readability of your code. But always be aware of possible design flaws you probably try to hide with them.

Everything about our Neos UI can be found at *https://docs.neos.io/cms/contributing-to-neos/neos-ui <https://docs.neos.io/cms/contributing-to-neos/neos-ui>*

Note: This is a documentation stub.

---

[1] This is also referred to as a bad "smell" in the theory of Refactoring. We highly recommend reading "Refactoring" by Martin Fowler - if you didn't already.

1.12 Configuration Reference

1.12.1 Navigation tree `loadingDepth`

`loadingDepth` defines the number of levels inside the node tree which shall be loaded eagerly, at start. A similar setting is available for the structure tree.

If you have lots of nodes you can reduce the number of levels inside `Settings.yaml` to speed up page loading:

```
Neos:
  Neos:
    userInterface:
      navigateComponent:
        nodeTree:
          loadingDepth: 2
        structureTree:
          loadingDepth: 2
```

1.12.2 Node tree presets

By default all node types that extend `Neos.Neos:Document` appear in the `Node tree filter` allowing the editor to only show nodes of the selected type in the tree.

The default `baseNodeType` can be changed in order to hide nodes from the tree by default.

This example shows how to exclude one specific node type (and it's children) from the tree:

```
Neos:
  Neos:
    userInterface:
      navigateComponent:
        nodeTree:
          presets:
            'default':
              baseNodeType: 'Neos.Neos:Document,!Acme.Com:SomeNodeTypeToIgnore'
```

In addition to the `default` preset, additional presets can be configured such as:

```
Neos:
  Neos:
    userInterface:
      navigateComponent:
        nodeTree:
          presets:
            'default':
              baseNodeType: 'Neos.Neos:Document,!Acme.Com:Mixin.HideInBackendByDefault'
            'legalPages':
              ui:
                label: 'Legal pages'
                icon: 'icon-gavel'
              baseNodeType: 'Acme.Com:Document.Imprint,Acme.Com:Document.Terms'
            'landingPages':
              ui:
```

(continues on next page)

(continued from previous page)

```
            label: 'Landing pages'
            icon: 'icon-bullseye'
          baseNodeType: 'Acme.Com:Mixin.LandingPage'
```

If at least one custom preset is defined, instead of the list of all node types the filter will display the configured presets.

1.12.3 Login Form

The login form can be customized by configuring a different background image or logo. You can even use custom CSS to change the look and feel of the login form.

To achieve this, you are able to change several values in your `Settings.yaml`: By default we use the following values:

```
Neos:
  Neos:
    userInterface:
      backendLoginForm:
          backgroundImage: 'resource://Neos.Neos/Public/Images/Login/Wallpaper.webp'
          logoImage: 'resource://Neos.Neos/Public/Images/Login/Logo.svg'
          stylesheets:
              'Neos.Neos:DefaultStyles': 'resource://Neos.Neos/Public/Styles/Login.css'
```

`backgroundImage` defines the background image of the login form on the login page. The image can be a resource or a path to an image.

`logoImage` defines the logo image of the login form on the login page. The image can be a resource or a path to an image.

`stylesheets` defines the stylesheets which are loaded on the login page.

1.13 Node Migration Reference

Node migrations can be used to deal with renamed node types and property names, set missing default values for properties, adjust content dimensions and more.

Node migrations work by applying **transformations** on nodes. The nodes that will be transformed are selected through **filters** in migration files.

The Content Repository comes with a number of common transformations:

- `AddDimensions`
- `AddNewProperty`
- `ChangeNodeType`
- `ChangePropertyValue`
- `RemoveNode`
- `RemoveProperty`
- `RenameDimension`
- `RenameNode`
- `RenameProperty`

- SetDimensions
- StripTagsOnProperty

They all implement the Neos\ContentRepository\Migration\Transformations\TransformationInterface. Custom transformations can be developed against that interface as well, just use the fully qualified class name for those when specifying which transformation to use.

1.13.1 Migration files

To use node migrations to adjust a setup to changed configuration, a YAML file is created that configures the migration by setting up filters to select what nodes are being worked on by transformations. The Content Repository comes with a number of filters:

- DimensionValues
- IsRemoved
- NodeName
- NodeType
- PropertyNotEmpty
- PropertyValue
- Workspace

They all implement the Neos\ContentRepository\Migration\Filters\FilterInterface. Custom filters can be developed against that interface as well, just use the fully qualified class name for those when specifying which filter to use.

Here is an example of a migration, Version20140708120530.yaml, that operates on nodes in the "live" workspace that are marked as removed and applies the RemoveNode transformation on them:

```
up:
  comments: 'Delete removed nodes that were published to "live" workspace'
  warnings: 'There is no way of reverting this migration since the nodes will be deleted␣
↪in the database.'
  migration:
    -
      filters:
        -
          type: 'IsRemoved'
          settings: []
        -
          type: 'Workspace'
          settings:
            workspaceName: 'live'
      transformations:
        -
          type: 'RemoveNode'
          settings: []

down:
  comments: 'No down migration available'
```

Like all migrations the file should be placed in a package inside the Migrations/ContentRepository folder where it will be picked up by the CLI tools provided with the content repository:

- `./flow node:migrationstatus`
- `./flow node:migrate`

Use `./flow help <command>` to get detailed instructions. The `migrationstatus` command also prints a short description for each migration.

Note: Node migrations in `Migrations/TYPO3CR` directories are also supported for historic reasons

---

1.13.2 Transformations Reference

AddDimensions

Add dimensions on a node. This adds to the existing dimensions, if you need to overwrite existing dimensions, use SetDimensions.

Options Reference:

`dimensionValues` (array)
An array of dimension names and values to set.

`addDefaultDimensionValues` (boolean)
Whether to add the default dimension values for all dimensions that were not given.

AddNewProperty

Add a new property with the given value.

Options Reference:

`newPropertyName` (string)
The name of the new property to be added.

`value` (mixed)
Property value to be set.

ChangeNodeType

Change the node type.

Options Reference:

`newType` (string)
The new Node Type to use as a string.

ChangePropertyValue

Change the value of a given property.

This can apply two transformations:

- If newValue is set, the value will be set to this, with any occurrences of the `currentValuePlaceholder` replaced with the current value of the property.
- If search and replace are given, that replacement will be done on the value (after applying the `newValue`, if set).

This would simply override the existing value:

```
transformations:
  -
    type: 'ChangePropertyValue'
    settings:
      property: 'title'
      newValue: 'a new value'
```

This would prefix the existing value:

```
transformations:
  -
    type: 'ChangePropertyValue'
    settings:
      property: 'title'
      newValue: 'this is a prefix to {current}'
```

This would prefix existing value and then apply search/replace on the result:

```
transformations:
  -
    type: 'ChangePropertyValue'
    settings:
      property: 'title'
      newValue: 'this is a prefix to {current}'
      search: 'something'
      replace: 'something else'
```

And in case your value contains the magic string "{current}" and you need to leav it intact, this would prefix the existing value but use a different placeholder:

```
transformations:
  -
    type: 'ChangePropertyValue'
    settings:
      property: 'title'
      newValue: 'this is a prefix to {__my_unique_placeholder}'
      currentValuePlaceholder: '__my_unique_placeholder'
```

Options Reference:

property (string)
: The name of the property to change.

newValue (string)
: New property value to be set.

The value of the option `currentValuePlaceholder` (defaults to "{current}") will be used to include the current property value into the new value.

`search` (string)
Search string to replace in current property value.

`replace` (string)
Replacement for the search string.

`currentValuePlaceholder` (string)
The value of this option (defaults to `{current}`) will be used to include the current property value into the new value.

RemoveNode

Removes the node.

RemoveProperty

Remove the property.

Options Reference:

`property` (string)
The name of the property to be removed.

RenameDimension

Rename a dimension.

Options Reference:

`newDimensionName` (string)
The new name for the dimension.

`oldDimensionName` (string)
The old name of the dimension to rename.

RenameNode

Rename a node.

Options Reference:

`newName` (string)
The new name for the node.

RenameProperty

Rename a given property.

Options Reference:

`from` (string)
The name of the property to change.

`to` (string)
The new name for the property to change.

SetDimensions

Set dimensions on a node. This always overwrites existing dimensions, if you need to add to existing dimensions, use AddDimensions.

Options Reference:

`dimensionValues` (array)
An array of dimension names and values to set.

`addDefaultDimensionValues` (boolean)
Whether to add the default dimension values for all dimensions that were not given.

StripTagsOnProperty

Strip all tags on a given property.

Options Reference:

`property` (string)
The name of the property to work on.

1.13.3 Filters Reference

DimensionValues

Filter nodes by their dimensions.

Options Reference:

`dimensionValues` (array)
The array of dimension values to filter for.

`filterForDefaultDimensionValues` (boolean)
Overrides the given dimensionValues with dimension defaults.

IsRemoved

Selects nodes marked as removed.

NodeName

Selects nodes with the given name.

Options Reference:

nodeName (string)
The value to compare the node name against, strict equality is checked.

NodeType

Selects nodes by node type.

Options Reference:

nodeType (string)
The node type name to match on.

withSubTypes (boolean)
Whether the filter should match also on all subtypes of the configured node type. Note: This can only be used with node types still available in the system!

exclude (boolean)
Whether the filter should exclude the given NodeType instead of including only this node type.

PropertyNotEmpty

Filter nodes having the given property and its value not empty.

Options Reference:

propertyName (string)
The property name to be checked for non-empty value.

PropertyValue

Filter nodes having the given property with the corresponding value.

Options Reference:

propertyName (string)
The property name to filter for with the given property value.

propertyValue (string)
The property value to filter for.

Workspace

Filter nodes by workspace name.

Options Reference:

`workspaceName` (string)
The workspace name to match on.

CHAPTER

TWO

CONTRIBUTE

2.1 UI Development

Warning: This applies for the legacy Ember Neos UI. Learn more about our **`current ReactJS UI in the docs<https://docs.neos.io/cms/contributing-to-neos/neos-ui>`_**.

2.1.1 Legacy Ember UI - Development

Warning: This applies for the legacy Ember Neos UI. Learn more about our current ReactJS UI in the docs. The EmberUI has been partially removed since Neos 5.0 and the last pieces (Notification API and Translation API) will follow soon!

Setting up your machine for Neos UI development

For user interface development of Neos we utilize *grunt* and some other tools.

Setting up your machine could be done by using the installation script that can be found in `Neos.Neos/Scripts/install-dev-tools.sh`. If you want to do a manual installation you will need to install the following software:

- nodejs
- npm
- grunt-cli (global, `sudo npm install -g grunt-cli`)
- requirejs (`sudo npm install -g requirejs`)
- bower (`sudo npm install -g bower`)
- bundler (`sudo gem install bundler`)
- sass & compass (`sudo gem install sass compass`)

Note: Make sure you call `npm install`, `bundle install --binstubs --path bundle` and `bower install` before running the grunt tasks.

Grunt tasks types

We have different types of grunt tasks. All tasks have different purposes:

- build commands

  Those commands are used to package a production version of the code. Like for example minified javascript, minified css or rendered documentation.

- compile commands

  Those commands are meant for compiling resources that are used in development context. This could for example be a packed file containing jquery and related plugins which are loaded in development context using requirejs.

- watch commands

  Those commands are used for watching file changes. When a change is detected the compile commands for development are executed. Use those commands during your daily work for a fast development experience.

- test commmands

  Used for running automated tests. Those tests use phantomjs which is automatically installed by calling `npm install`. Phantomjs needs some other dependencies though, check `Neos.Neos/Scripts/install-phantomjs-dependencies.sh` for ubuntu based systems.

Available grunt tasks

Build

- `grunt build`

  Executes `grunt build-js` and `grunt build-css`.

- `grunt build-js`

  Builds the minified and concatenated javascript sources to `ContentModule-built.js` using requirejs.

- `grunt build-css`

  Compiles and concatenates the css sources to `Includes-built.css`.

- `grunt build-docs`

  Renders the documentation. This task depends on a local installation of Omnigraffle.

Compile

- `grunt compile`

  Executes `grunt compile-js` and `grunt compile-css`

- `grunt compile-js`

  Compiles the javascript sources. This is the task to use if you want to package the jquery sources including plugins or if you want to recreated the wrapped libraries we include in Neos. During this process some of the included libraries are altered to prevent collisions with Neos or the website frontend.

- `grunt compile-css`

  Compiles and concatenates the scss sources to css.

Watch

- `watch-css`

  Watches changes to the scss files and runs `compile-css` if a change is detected.

- `watch-docs`

  Watches changes to the rst files of the documentation, and executes a compilation of all restructured text sources to html. This task depends on a local sphinx install but does not require Omnigraffle.

- `watch`

  All of the above.

Test

- `grunt test`

  Runs QUnit tests for javascript modules.

2.2 API Documentation

Our documentation is managed in a Neos instance at https://docs.neos.io/

This references and versioned content is build with Sphinx. You can learn more about how this is build here:

2.2.1 Neos Documentation

How it works

Our documentation is managed in a Neos instance at https://docs.neos.io/

We use Read The Docs (http://neos.readthedocs.org) to host the versionized API documentation for Neos.

This service listens for commits on Github and automatically builds the documentation for all branches.

The entire documentation of Neos is located inside the Neos development collection (https://github.com/neos/neos-development-collection) and can be edited by forking the repository, editing the files and creating a pull request.

reStructuredText

The markup language that is used by Sphinx is [reStructuredText](http://docutils.sourceforge.net/rst.html), a plaintext markup syntax that easy to edit using any text editor and provides the possibility to write well organized documentations that can be rendered in multiple output formats by e.g. Sphinx.

Sphinx

Sphinx is a generator that automates building documentations from reStructuredText markup. It can produce HTML, LaTex, ePub, plain text and many more output formats.

As Sphinx is a python based tool, you can install it by using either pip:

```
pip install -U Sphinx
```

or easy_install:

```
easy_install -U Sphinx
```

Makefile

As Sphinx accepts many options to build the many output formats, we included a *Makefile* to simplify the building process.

In order to use the commands you must already have Sphinx installed.

You can get an overview of the provided commands by

```
cd Neos.Neos/Documentation
```

```
make help
```

Docker

If you don't want to install Sphinx on your computer or have trouble installing it, you can use a prebuilt Docker image that contains a working version of Sphinx. The image is built on top of a pretty small alpine linux and has only around 80MB.

You can simply prefix your *make* command with the following docker command:

```
docker run -v $(pwd):/documents hhoechtl/doctools-sphinx make html
```

This will fire up a docker-container built from that image and execute the Sphinx build inside the container. As your current directory is mounted into the container, it can read the files and the generated output will be written in your local filesystem as it would by just executing the make command with your local Sphinx installation.

2.2.2 Beginners Guide Sphinx-Setup

Contribute to the Neos-Documentation

This Documentation aims to get you started quite from the ground up. A lot of explainations here can of cause be used to work on the whole repository, it just seems to be a good starting point to explain the workflow concerning the documentation first.

Imagine you would like to contribute to the Documentation but you haven't worked with github yet, you don't know how a proper workflow looks like and you are not sure how to start contributing. The problem is, that even while explaining some of the basic steps, there always is the need for some kind of basic setup you will have to take care of yourself. You can of cause commit by using GitHub itself. The aim of this document is focusing on working with git locally. You need for eg. a Linux Console and git to get started.[1]

[1] The basic setup, this Tutorial and the Screenshots are based on Arch Linux, Awesome (as a Window Manager), bash (with urxvt) and ice-firefox (the single-page-browser ice-spb) and Atom as the Editor.

What are the goals?

Once everything is set up nicely and hopefully without to much trouble, you will:

- know how to commit changes directly on GitHub.
- be able to easily access the Documentation offline in your browser
- know how to work with git and hub effectively when editing the Documentation
- see the life updated changes in your browser
- send pull request for your changes back to the Neos-Team
- see how to do some basic formatting with reStructuredText (rST)
- know how to use the todo functionality

Let's get started

The easiest way to start is using GitHubs website itself to work on the repository. Just click on the fork-button inside the repository, once you have done this you have got your own copy (fork) of the repo you can work on. At first create a new branch by clicking on the branch-button and typing in a new appropriate branch-name into the input field.

Next you can start editing the files relating to the branch you just created. Now you just need to save your changes by clicking the "Commit changes"-button. (Please read the part below about meaningful commit messages).

Once you have done all the necessary changes you can click the "Create pull request"-button. Again make sure to explain what you have done. This last step opens also a new dialog about your pull request in the original forked repository. Depending of what you have done this will either be merged right away or you might get some feedback if some work might still be necessary.

That's basically it. Next we will look into the way of making your commits more precise before discussing a detailed offline way of working on the repository.

Guideline - commit messages

Note: The following section was originally posted here (commit message style) by Christian Müller. Please make sure to follow these Guidelines.

To have a clear and focused history of code changes is greatly helped by using a consistent way of writing commit messages. Because of this and to help with (partly) automated generation of change logs for each release we have defined a fixed syntax for commit messages that is to be used.

> **Warning:** Tip: Never commit without a commit message explaining the commit
>
> The syntax is as follows:

Start with one of the following codes:

Note: FEATURE A feature change. Most likely it will be an added feature, but it could also be removed. There should be a corresponding ticket in the issue tracker. Features usually only get into the current development master.

BUGFIX A fix for a bug. There should be a ticket corresponding to this in the issue tracker and we encourage to add a new test that exposes the bug, which makes the work for everyone easier in the future and prevents the bug from reappearing.

TASK Anything not covered by the above categories, e.g. coding style cleanup or documentation changes. Usually only used if there's no corresponding ticket.

SECURITY A security related change. Those are only commited by active team members in the security community of practice.

MERGE Used for a branch upmerges by the team (or CI server) not something you usually would need to use.

The code is separated by a colon : from a short summary in the same line, no full stop at the end.

If the change affects the public API or is likely to break things on the user side, prefix the line with !!!. This indicates a breaking change that needs human action when updating. Make sure to explain why a change is breaking and in what circumstances. A change including a migration should always be marked breaking to alert users of the need to migrate.

Then (after a blank line) follows the custom message explaining what was done. It should be written in a style that serves well for a change log read by users. If there is more to say about a change add a new paragraph with background information below. In case of breaking changes give a hint on what needs to be changed by the user. If corresponding tickets exist, mention the ticket number(s) using footer lines after another blank line and use the following actions:

<issue number> #close Some additional info if needed If the change resolves a ticket by fixing a bug, implemeting a feature or doing a task. <issue number> #comment Some info why this is related If the change relates to an issue but does not resolve or fix it. This follows Jiras smart commit footers, see more details in the Jira documentation3

A commit messages following the rules. . . :

Note: TASK: Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary. Wrap it to about 72 characters or so. In some contexts, the first line is treated as the subject of an email and the rest of the text as the body. The blank line separating the summary from the body is critical (unless you omit the body entirely); tools like rebase can get confused if you run the two together.

Write your commit message in the present tense: "Fix bug" and not "Fixed bug." This convention matches up with commit messages generated by commands like git merge and git revert.

Code snippets:

```
should be written in
ReStructuredText compatible
format for better highlighting
```

Further paragraphs come after blank lines.

- Bullet points are okay, too
- An asterisk is used for the bullet, it can be preceded by a single space. This format is rendered correctly by Forge (redmine)
- Use a hanging indent

A first step in solving neos/flow-development-collection#789.

Fixes #123

Closes #456

Examples of good and bad subject lines:

Note: Introduce xyz service BAD, missing code prefix

BUGFIX: Fixed bug xyz BAD, subject should be written in present tense

TASK!!!: A breaking change BAD, subject has to start with !!! for breaking changes

BUGFIX: SessionManager removes expired sessions GOOD, the line explains what the change does, not what the bug is about (this should be explained in the following lines and in the related bug tracker ticket)

!!! BUGFIX: SessionManager never expires sessions GOOD, the line explains what the change does, not what the bug is about (this should be explained in the following lines and in the related bug tracker ticket)

Warning: Please also have a look at this discussion: (Creating a pull request).

Using git in the console

```
sudo apt-get install git-all hub #(Debian Based)
sudo pacman -Sy git hub #(Arch Linux)
```

Quote:

> "Whether you are beginner or an experienced contributor to open-source, hub makes it easier to fetch repositories, navigate project pages, fork repos and even submit pull requests, all from the command-line." – hub.github.com

The Atom Editor including the extension packages *Git Diff* and *language-restructuredtext* would be nice options for editing the files, etc…:

```
yaourt atom-editor #(Arch Linux)
```

(See https://github.com/atom/atom for other Distributions)[2]

Here you can see how the Atom Editor looks like. On the left side you can see, that the new (green) and changed (yellow) folders and files are highlighted, also in the document itself you can see which lines you changed or added:

To be able to work with GitHub nicely from the console, you could use hub instead of git, for that you can edit and add: *alias git=hub* to the .bashrc and refresh it:

```
vim ~/.bashrc #(add: alias git=hub)
source ~/.bashrc #(to reload the .bashrc-file)
```

The Neos Development Collection Repository

Now lets clone the Neos Development Collection Repository into the folder you are currently in.

```
git clone https://github.com/neos/neos-development-collection.git
```

[2] The Atom Editor is just one example of many good Editors out there, also the given Information here might not be enough the Arch Linux command makes necessary to have set up AUR and yaourt otherwise you won't be able to run that command at all…

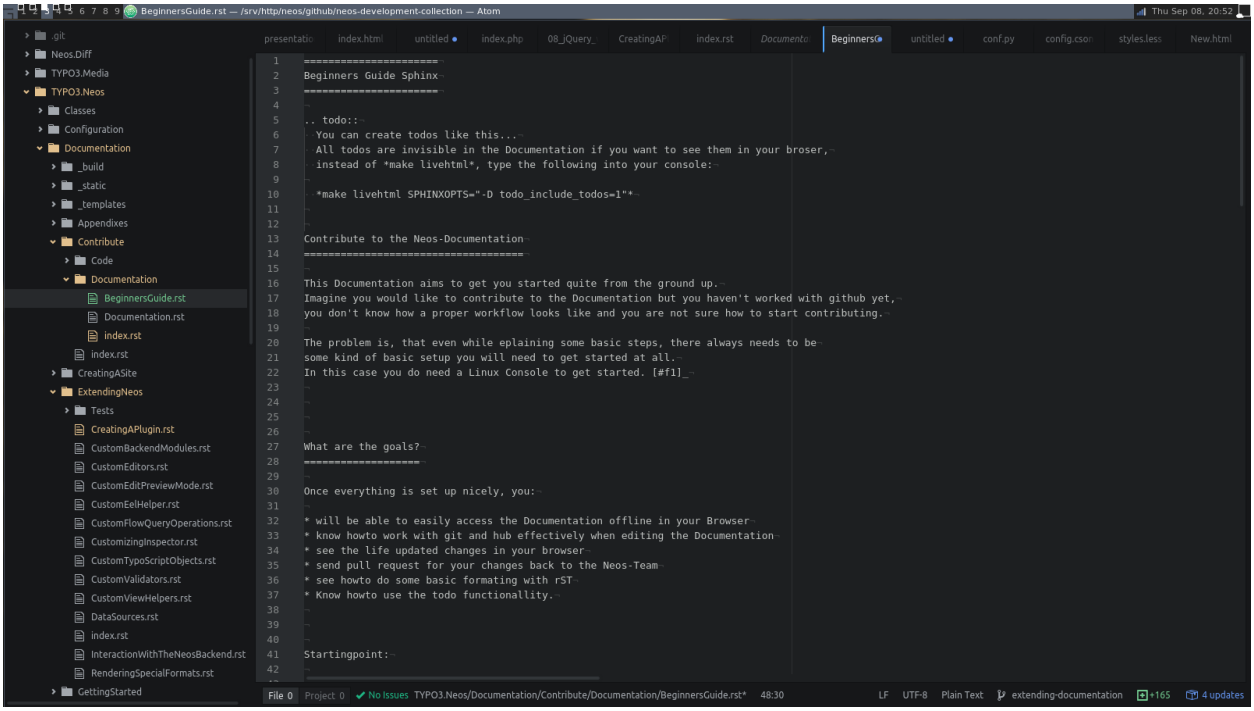


Fig. 1: The Atom Editor

Sphinx requirements

Sphinx is based on Python to make Sphinx available in your System you probably need to install some packages through pip.

```
sudo pacman -S python-pip
```

There are different ways of dealing with Python-packages. The following way is to install it in the user-directory and a dding the bin-path to the $PATH – Environment.

```
pip install --user Sphinx
pip install --user sphinx-autobuild
pip install --user sphinx_rtd_theme
```

Then add the following line to your .bashrc: *export PATH=$HOME/.local/bin:$PATH*

```
vim ~/.bashrc #(add the above line)
source ~/.bashrc #(to reload the .bashrc-file)
```

Let the fun begin

Now you should already be able to make the documentation available in the browser. Go into the following folder from where you cloned the Neos-Collection:

cd /neos-development-collection/Neos.Neos/Documentation/

And then run the following command:

```
make livehtml
```

If everything works as planed, you should now see a line like this in the console:

[I 160908 18:55:04 server:281] Serving on http://127.0.0.1:8000

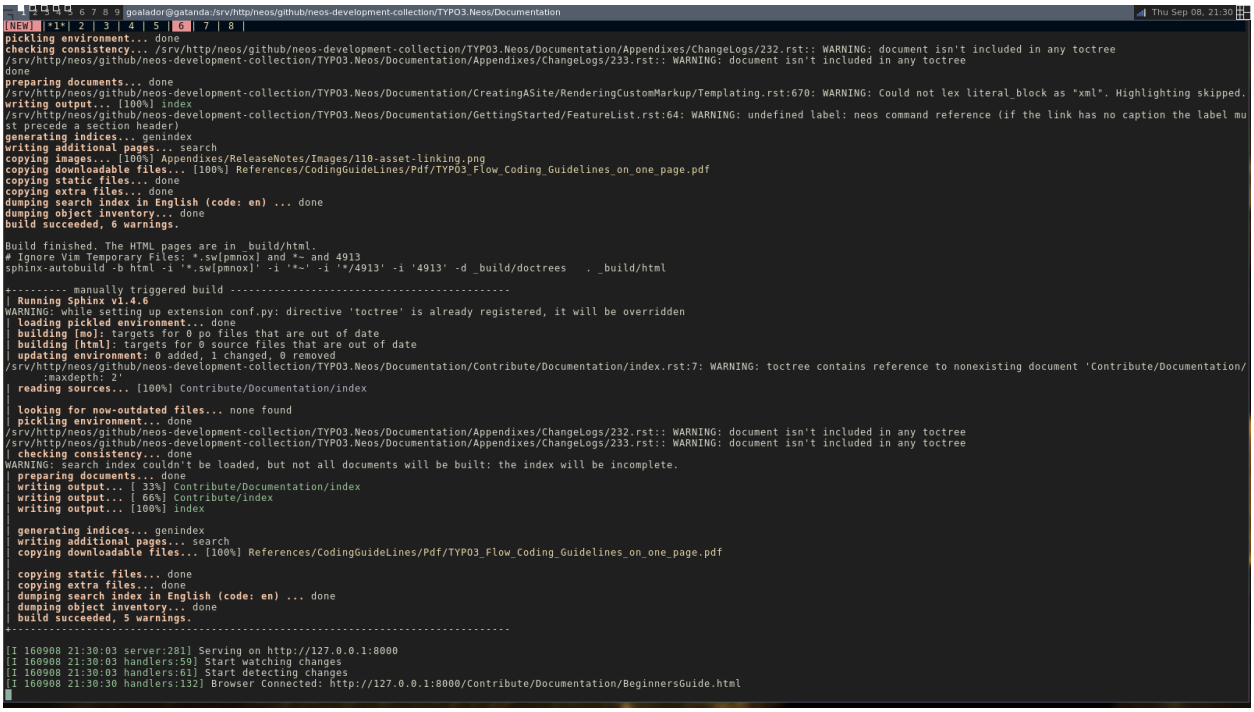


Fig. 2: Sphinx make livehtml

The Url served here is, as long as you keep the process running, live reloaded when the files are changed. Just open the Url in your Browser, you should see the whole Documentation served by your local machine. Now try to open a file in the Neos-Collection eg. the file you are reading right now is located here: */neos-development-collection/Neos.Neos/Documentation/Contribute/Documentation/BeginnersGuide.rst*

Now change a line, save it and have a look in the console and the browser. Afterwards undo the change, to make sure git doesn't take the change seriously yet… The console should have recognised by now, that you are connected with a browser to the url, and now should also tell you which file was changed. If you check the browser again, it should, without manually refreshing the page, show you the edited line in its new version.

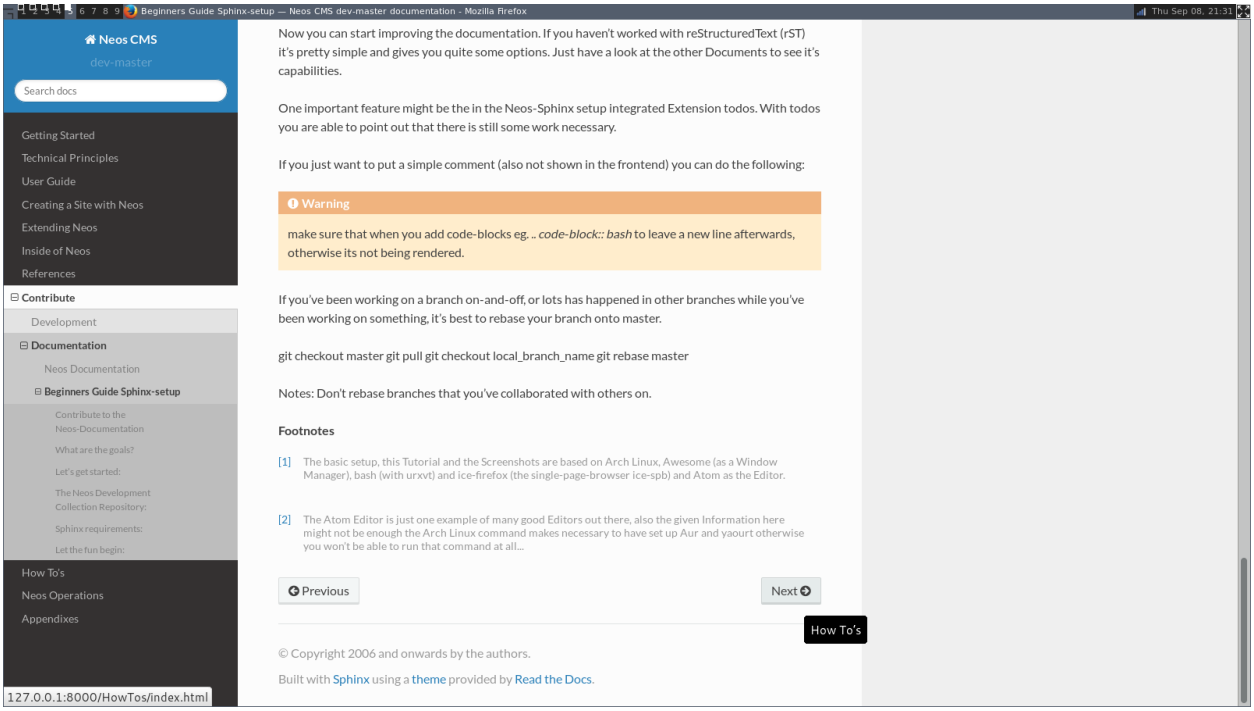


Fig. 3: Sphinx browser view

reStructuredText (rST)

Now you can start improving the documentation. If you haven't worked with reStructuredText (rST) it's pretty simple and gives you quite some options. Just have a look at the Documentation files available, they give you a good understanding of what is possible. It has a lot of capabilities. Checkout their documentation for more informations Sphinx docs.

One nice feature is the, in the Neos-Sphinx setup integrated, extension *todo*. With *todo* you are able to point out that there is still some work necessary. Add a todo, if you feel like there is something missing here, or someone else needs to check if what you have written is correct like this. Just use it a lot to make sure it's obvious what still needs to be done...

Note: Every following line which is indented by two spaces now, is part of the note. If you would replace it with todo instead of (.. *note::* -> .. *todo::*), it wouldn't be visible in the frontend/browser anymore, but just just visible for you and others, when editing these files.

There is also the possibility to see all the todos with its positions by putting .. *todolist::* into the document. Both features (the todo itself and their collection) can be made visible in the browser while working on the documentation for eg. by starting Sphinx like this:

```
make livehtml SPHINXOPTS="-D todo_include_todos=1"
```

If you just want to put a simple comment (also not shown in the frontend) you can do the following:

Note: Comments are also invisible in the browser, you can create them by just using two dots (..) at the beginning of a line. The following indented lines are part of the comment.

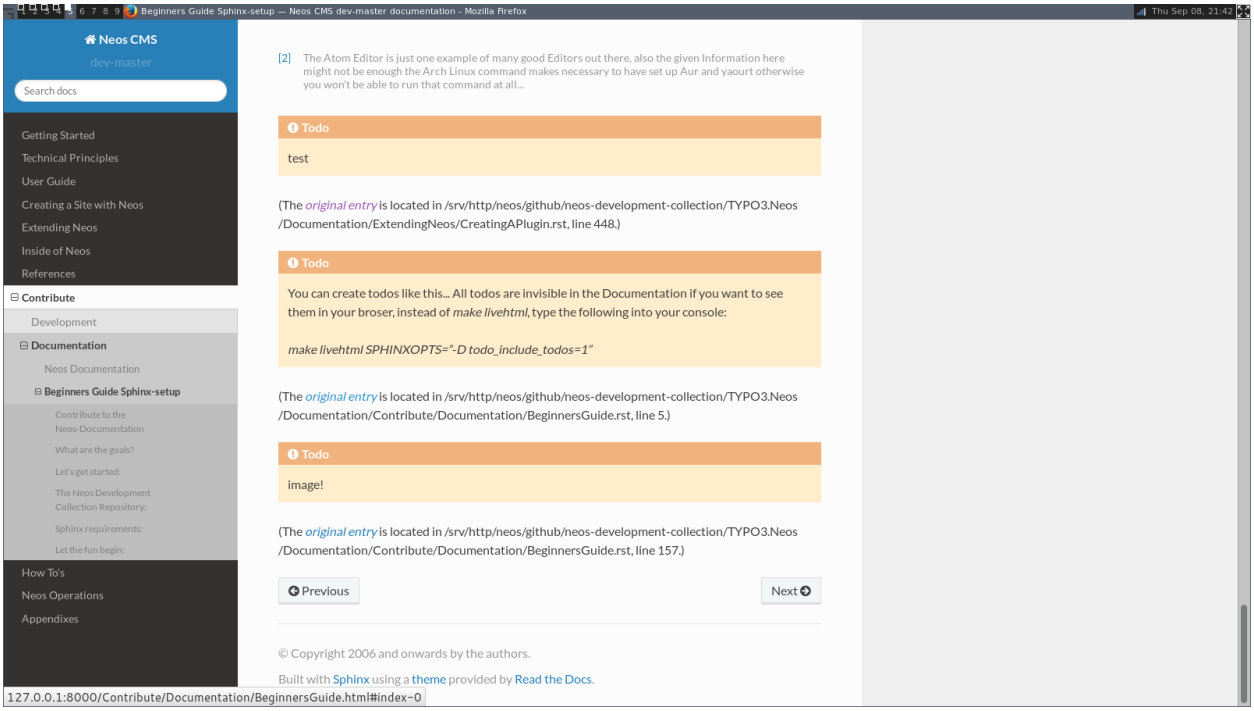


Fig. 4: Sphinx todolist

Warning: Make sure that when you add code-blocks eg. .. *code-block:: bash* to leave a new line afterwards, otherwise its not being rendered.

GitHub checkout-process

Now we should have a look at the git-workflow. The first step you should checkout a branch from master to be able to work on that locally for now. Somewhere below the Folder *neos-development-collection/*, you should run the following command to create and enter a new branch:

```
git checkout -b [local_branch_name]
```

Now you can start editing the files as you like in your own local feature-branch.

If you've been working on a branch here and there, you should probably make sure first, that your master-branch is up to date. The there are two strategies for that. Here we will rebase your only local branch onto master. The following would be an example where you stash your changes for now, so you don't have to commit them there and then, switch to your local master, pull the changes to be up to date and then apply your changes back to your reactivated feature-branch.

```
git stash
git checkout master
git pull
git checkout [local_branch_name]
git rebase master
git stash apply
```

Warning: Make sure not to rebase branches that you've collaborated with others on. Never rebase anything you have pushed somewhere already.

To get more information about how to work with git go to this page there are many good sources online. Two good examples are for eg.: SSH, Basic Branching and Merging or also Rebasing.

```
git add [new files]
git commit -m "FEATURE done with the feature: [local_branch_name] to make this and that␣
↪more efficient"
git fork #(forking repo on GitHub...)
#→ git remote add YOUR_USER git://github.com/YOUR_USER/neos-development-collection.git
```

```
# push the changes to your new remote
git push YOUR_USER feature
# open a pull request for the topic branch you've just pushed
git pull-request
#→ (opens a text editor for your pull request message)
```

CHAPTER

THREE

NEOS OPERATIONS

3.1 Command Line Tools

Neos comes with a number of command line tools to ease setup and maintenance. These tools can be used manually or be added to automated deployments or cron jobs. This section gives a high level overview of the available tools.

More detailed instructions on the use of the command line tools can be displayed using the `help` command:

```
./flow help                    # lists all available command
./flow help <packageKey>       # lists commands provided in package
./flow help <commandIdentifier> # show help for specific command
```

Here is an example:

```
./flow help user:addrole

Add a role to a user

COMMAND:
  neos.neos:user:addrole

USAGE:
  ./flow user:addrole [<options>] <username> <role>

ARGUMENTS:
  --username           The username of the user
  --role               Role to be added to the user, for example
                       "Neos.Neos:Administrator" or just "Administrator

OPTIONS:
  --authentication-provider Name of the authentication provider to use. Example:
                       "Neos.Neos:Backend

DESCRIPTION:
  This command allows for adding a specific role to an existing user.

  Roles can optionally be specified as a comma separated list. For all roles provided by␣
↪Neos, the role
  namespace "Neos.Neos:" can be omitted.

  If an authentication provider was specified, the user will be determined by an account␣
```

(continues on next page)

(continued from previous page)

```
↪identified by "username"
  related to the given provider. However, once a user has been found, the new role will␣
↪be added to all
  existing accounts related to that user, regardless of its authentication provider.
```

3.1.1 User Management

These commands allow to manage users. To create an user with administrative privileges, this is needed:

```
./flow user:create john@doe.com pazzw0rd John Doe --roles Neos.Neos:Administrator
```

Command	Description
user:list	List all users
user:show	Shows the given user
user:create	Create a new user
user:delete	Delete a user (with globbing)
user:activate	Activate a user (with globbing)
user:deactivate	Deactivate a user (with globbing)
user:setpassword	Set a new password for the given user
user:addrole	Add a role to a user (with globbing)
user:removerole	Remove a role from a user (with globbing)

3.1.2 Workspace Management

The commands to manage workspaces reflect what is possible in the Neos user interface. They allow to list, create and delete workspaces as well as publish and discard changes.

One notable difference is that rebasing a workspace is possible from the command line *even if it contains unpublished changes*.

Command	Description
workspace:publish	Publish changes of a workspace
workspace:discard	Discard changes in workspace
workspace:create	Create a new workspace
workspace:delete	Deletes a workspace
workspace:rebase	Rebase a workspace
workspace:list	Display a list of existing workspaces

3.1.3 Site Management

Command	Description
domain:add	Add a domain record
domain:list	Display a list of available domain records
domain:delete	Delete a domain record (with globbing)
domain:activate	Activate a domain record (with globbing)
domain:deactivate	Deactivate a domain record (with globbing)
site:import	Import sites content
site:export	Export sites content
site:prune	Remove all content and related data (with globbing)
site:list	Display a list of available sites

CHAPTER

FOUR

APPENDIXES

CHAPTER

FIVE

INDICES AND TABLES

- genindex