Neos CMS

Release 4.3.x

Neos Team and Contributors

Nov 08, 2021

CONTENTS

1 Getting Started **3**
- 1.1 Installation . . . 3

2 Technical Principles **11**

3 Creating a Site with Neos **13**
- 3.1 Node Types . . . 13
- 3.2 Fusion . . . 31
- 3.3 Rendering Custom Markup . . . 47
- 3.4 Content Dimensions . . . 81
- 3.5 Multi Site Support . . . 85
- 3.6 Content Cache . . . 85
- 3.7 Permissions & Access Management . . . 92

4 Extending Neos **105**
- 4.1 Creating a plugin . . . 105
- 4.2 Custom Backend Modules . . . 110
- 4.3 Custom Edit/Preview-Modes . . . 112
- 4.4 Custom Editors . . . 113
- 4.5 Custom Eel Helper . . . 116
- 4.6 Custom FlowQuery Operations . . . 117
- 4.7 Custom Fusion Objects . . . 119
- 4.8 Custom Validators . . . 120
- 4.9 Custom ViewHelpers . . . 120
- 4.10 Customizing the Inspector . . . 123
- 4.11 Data sources . . . 126
- 4.12 Interaction with the Neos backend . . . 128
- 4.13 Rendering special formats (CSV, JSON, XML, ...) . . . 129
- 4.14 Neos User Interface Extensibility API . . . 130
- 4.15 Writing Tests For Neos . . . 139

5 Inside of Neos **145**
- 5.1 User Interface Development . . . 145

6 References **157**
- 6.1 Property Editor Reference . . . 157
- 6.2 View Helper Reference . . . 169
- 6.3 Fusion Reference . . . 241
- 6.4 Eel Helpers Reference . . . 267
- 6.5 FlowQuery Operation Reference . . . 305
- 6.6 Neos Command Reference . . . 315

6.7 Validator Reference . . . 349
6.8 Signal Reference . . . 360
6.9 Coding Guideline Reference . . . 374
6.10 Configuration Reference . . . 400
6.11 Node Migration Reference . . . 401

7 Contribute **407**
7.1 Development . . . 407
7.2 Documentation . . . 409

8 How To's **419**
8.1 Neos Best Practices (to be written) . . . 419
8.2 Adding A Simple Contact Form . . . 419
8.3 Changing the Body Class with a condition . . . 421
8.4 Changing Defaults Depending on Content Placement . . . 421
8.5 Creating a simple Content Element . . . 422
8.6 Customize Login Screen . . . 423
8.7 Editing a shared footer across all pages . . . 424
8.8 Extending the Page . . . 425
8.9 Integrating a JavaScript-based slider . . . 426
8.10 Rendering Custom Document Types . . . 428
8.11 Rendering a Menu . . . 429
8.12 Rendering a Meta-Navigation . . . 430
8.13 Tagging assets automatically . . . 432
8.14 Translating content . . . 434
8.15 Wrapping a List of Content Elements . . . 435

9 Neos Operations **439**
9.1 Command Line Tools . . . 439

10 Appendixes **441**

11 Indices and tables **443**

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

This version of the documentation covering Neos 4.3.x has been rendered at: Nov 08, 2021

CHAPTER

ONE

GETTING STARTED

1.1 Installation

Tip: Neos is built on top of the Flow framework. If you run into technical problems, keep in mind to check the Flow documentation for possible hints as well.

1.1.1 Requirements

Neos has at least the same system requirements as Flow. You can find them in the Flow Requirements Documentation.

The most important requirements are:

- A Webserver (Apache and Nginx are preferred but others work as well)
- A Database (MySQL > 5.7.7, MariaDB > 10.2.2 and PostgreSQL > 9.4 are preferred but any Database supported by Doctrine DBAL should work as well).
- PHP >= 7.1.0 (make sure the PHP CLI has the same version)
  - PHP modules mbstring, tokenizer and pdo_mysql
  - PHP functions exec(), shell_exec(), escapeshellcmd() and escapeshellarg()
  - It is recommended to install one of the PHP modules imagick or gmagick

1.1.2 Fundamental Installation

1. First you need to install the dependency manager *Composer* (if you don't have it already):

   ```
   curl -sS https://getcomposer.org/installer | php
   ```

   By issuing this command Composer will get downloaded as *composer.phar* to your working directory. If you like to have composer installed globally, you can simply move it to a directory within your $PATH environment.

   ```
   mv composer.phar /usr/local/bin/composer
   ```

   Note: If you are on Windows please refer to the offical documentation on how to install Composer on Windows

2. Go to your htdocs directory and create a new project based on the Neos base distribution:

```
cd /your/htdocs/
php /path/to/composer.phar create-project neos/neos-base-distribution Neos
```

Composer will take care of downloading all dependencies for running your Neos installation to the directory `Neos`. You can safely delete the vcs files by answering 'Y' to the question 'Do you want to remove the existing VCS (.git, .svn..) history? [Y,n]?'.

3. Next set up a virtual domain/host in your webserver configuration

   - Apache configuration

     Set up a virtual host inside your Apache configuration. Set the `DocumentRoot` to the `Web` directory inside the Neos installation. Set the directive `AllowOverride` to `ÀLL` to allow access to *.htaccess*

     ```
     NameVirtualHost *:80 # if needed

     <VirtualHost *:80>
       DocumentRoot "/your/htdocs/Neos/Web/"
       # enable the following line for production context
       #SetEnv FLOW_CONTEXT Production
       ServerName neos.demo
     </VirtualHost>

     <Directory /your/htdocs/Neos/Web>
       AllowOverride All
     </Directory>
     ```

     Make sure that the `mod_rewrite` module is loaded and restart apache. For further information on how to set up a virtual host with apache please refer to the Apache Virtual Host documentation.

   - nginx configuration

     For further information on how to set up a virtual domain with nginx please refer to the nginx documentation.

4. Add an entry to */etc/hosts* to make your virtual host reachable:

   ```
   127.0.0.1 neos.demo
   ```

   Make sure to use the same name you defined in `ServerName` in the virtual host configuration above.

5. Set file permissions as needed so that the installation is read- and writeable by the webserver's user and group:

   ```
   sudo ./flow core:setfilepermissions john www-data www-data
   ```

   Replace *john* with your current username and *www-data* with the webserver's user and group.

   For detailed instructions on setting the needed permissions see Flow File Permissions

   ---

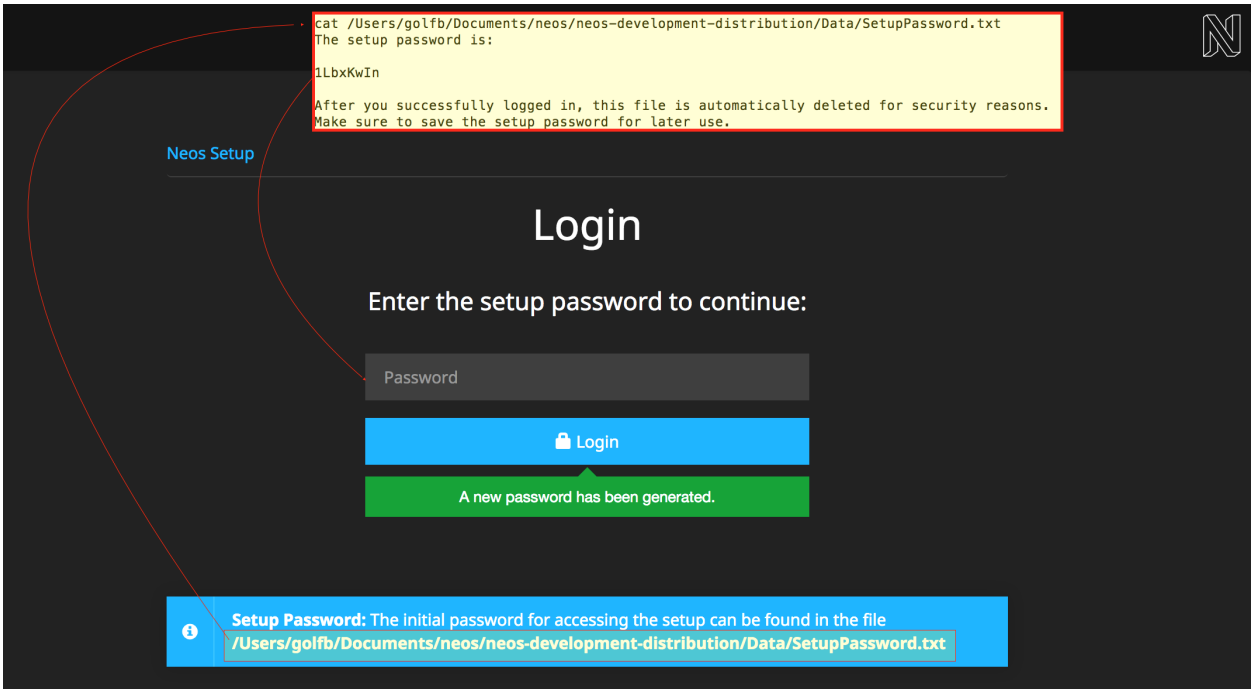
   Note: Setting file permissions is not necessary and not possible on Windows machines. For Apache to be able to create symlinks, you need to use Windows Vista (or newer) and Apache needs to be started with Administrator privileges.

   ---

6. Now go to http://neos.demo/setup and follow the on-screen instructions.

1.1.3 The Neos Setup Tool

1. A check for the basic requirements of Flow and Neos will be run. If all is well, you will see a login screen. If a check failed, hints on solving the issue will be shown and you should fix what needs to be fixed. Then just reload the page, until all requirements are met.
2. The login screen will tell you the location of a file with a generated password. Keep that password in some secure place, the generated file will be removed upon login! It is possible to have a new password rendered if you lost it, so don't worry too much.



3. The *NEOS requirements check* checks, if you have installed an image manipulation software.
4. Fill in the database credentials in the first step. The selector box will be updated with accessible databases to choose from, or you can create a new one.

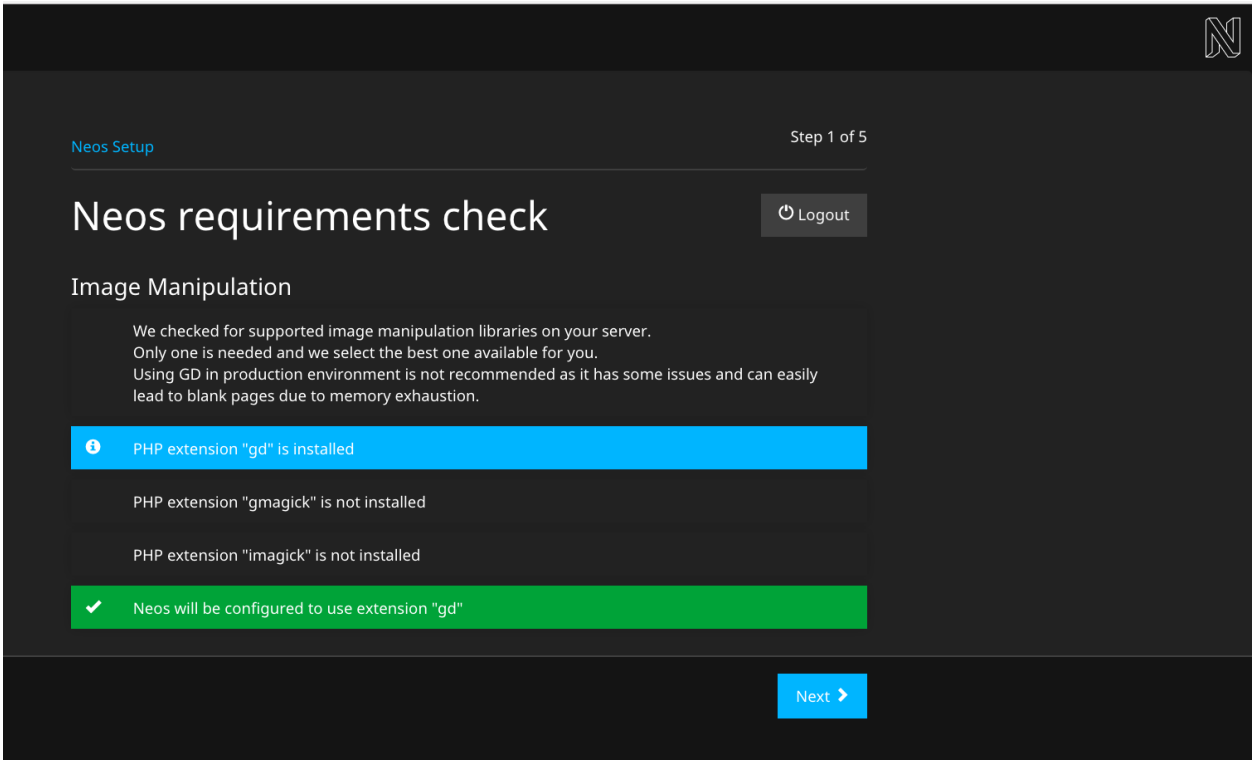
Tip: Configure your MySQL server to use the `utf8_unicode_ci` collation by default if possible!

---

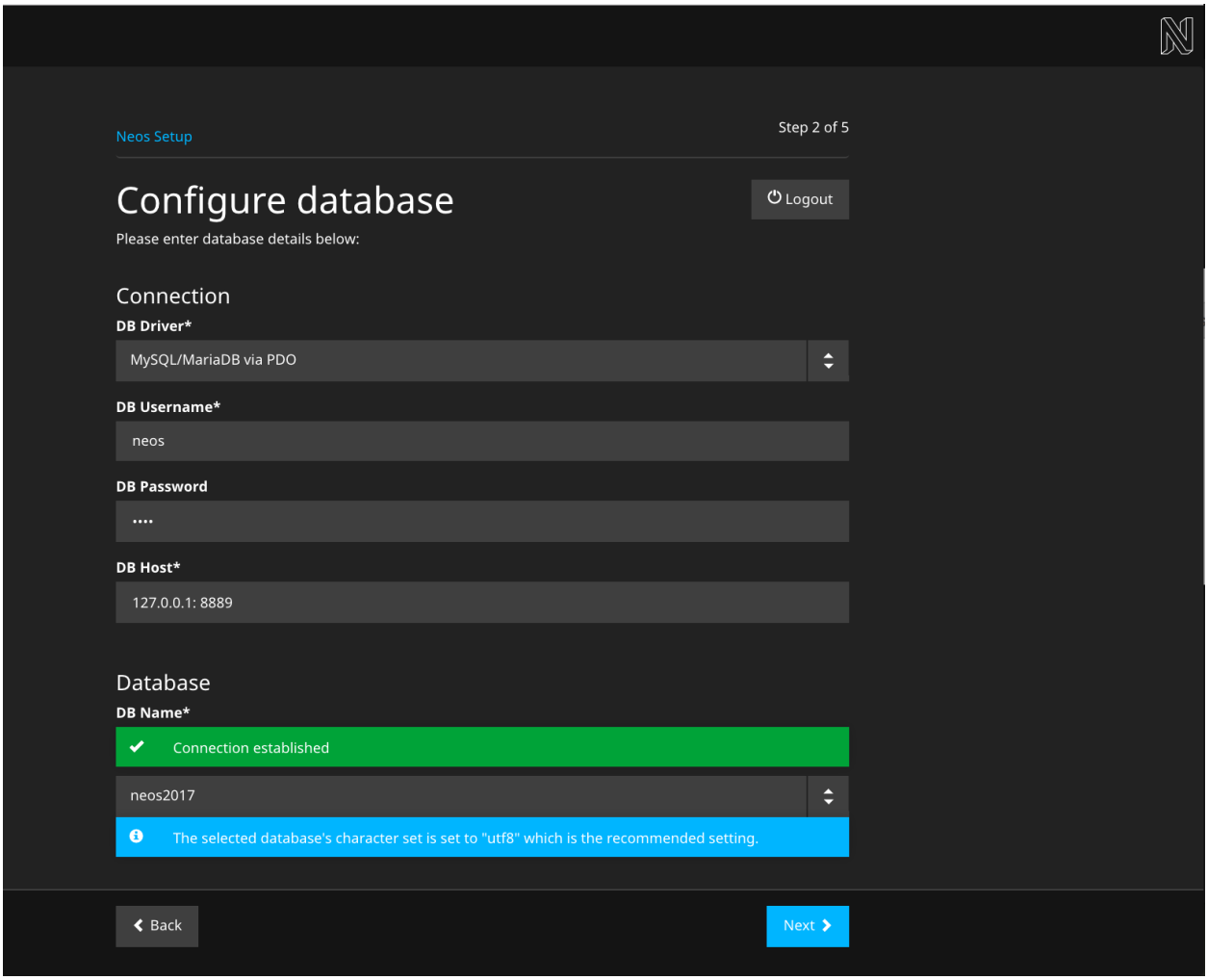
5. In the next step a user with administrator privileges for editing with Neos is created.
6. The following step allows you to import an existing site or kickstart a new site. To import the demo site, just make sure it is selected in the selector box and go to the next step.

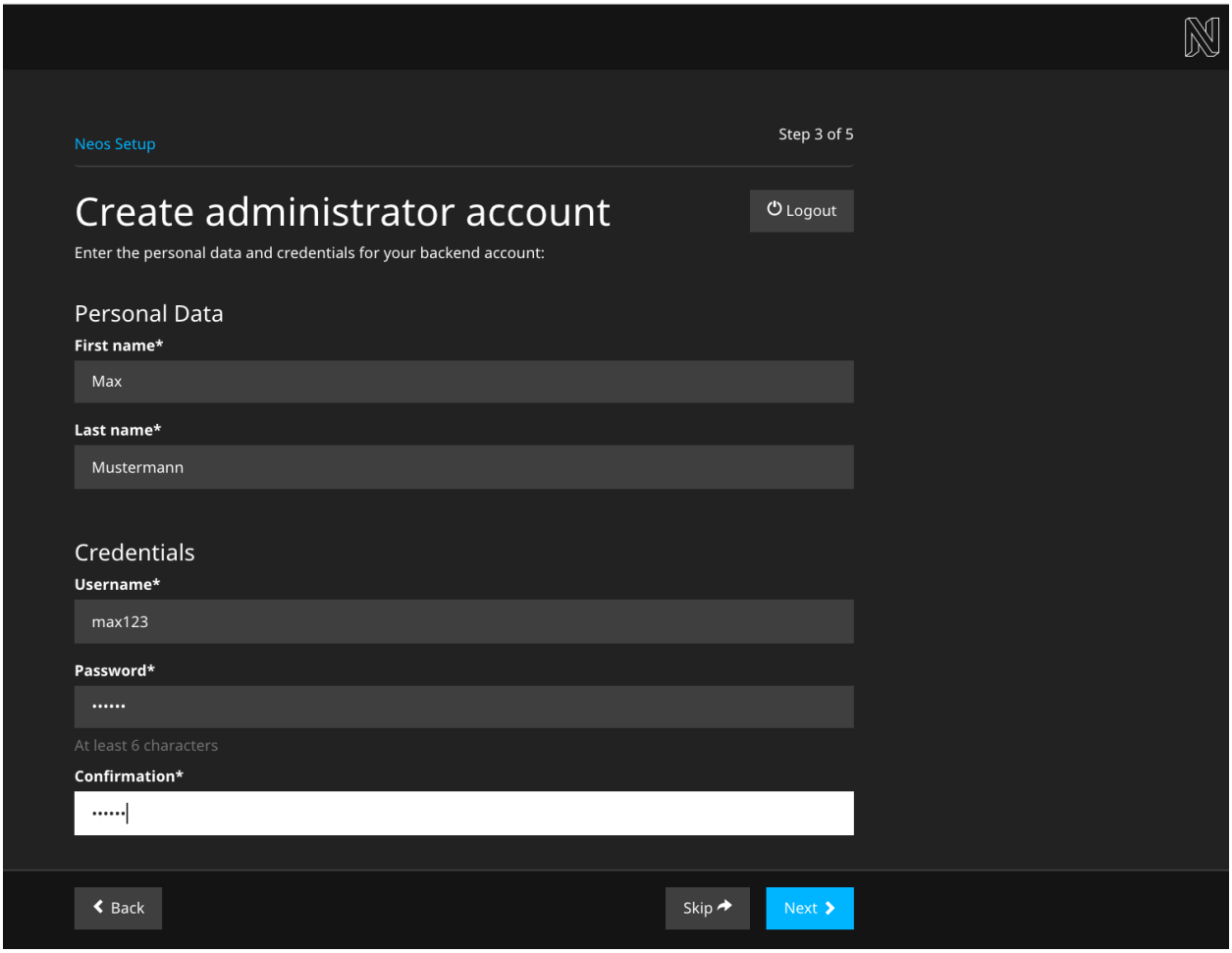
   To kickstart a new site, enter a package and site name in the form before going to the next step.

   If you are new to Neos, we recommend to import the existing demo site so you can follow the next section giving you a basic tour of the user interface.
7. If all went well you'll get a confirmation the setup is completed, and you can enter the frontend or backend of your Neos website.



Warning: If you install the Neos demo site and it is publicly accessible, make sure the “Try me” page in the page tree is not publicly accessible because it has a form allowing you to create backend editor accounts with rights to edit website content.)


Neos Setup
Step 2 of 5
Configure database
Logout
Please enter database details below:
Connection
DB Driver*
MySQL/MariaDB via PDO
DB Username*
neos
DB Password
••••
DB Host*
127.0.0.1: 8889
Database
DB Name*
Connection established
neos2017
The selected database's character set is set to "utf8" which is the recommended setting.
Back
Next


Neos Setup
Step 3 of 5
Create administrator account
Logout
Enter the personal data and credentials for your backend account:
Personal Data
First name*
Max
Last name*
Mustermann
Credentials
Username*
max123
Password*
At least 6 characters
Confirmation*
Back
Skip
Next

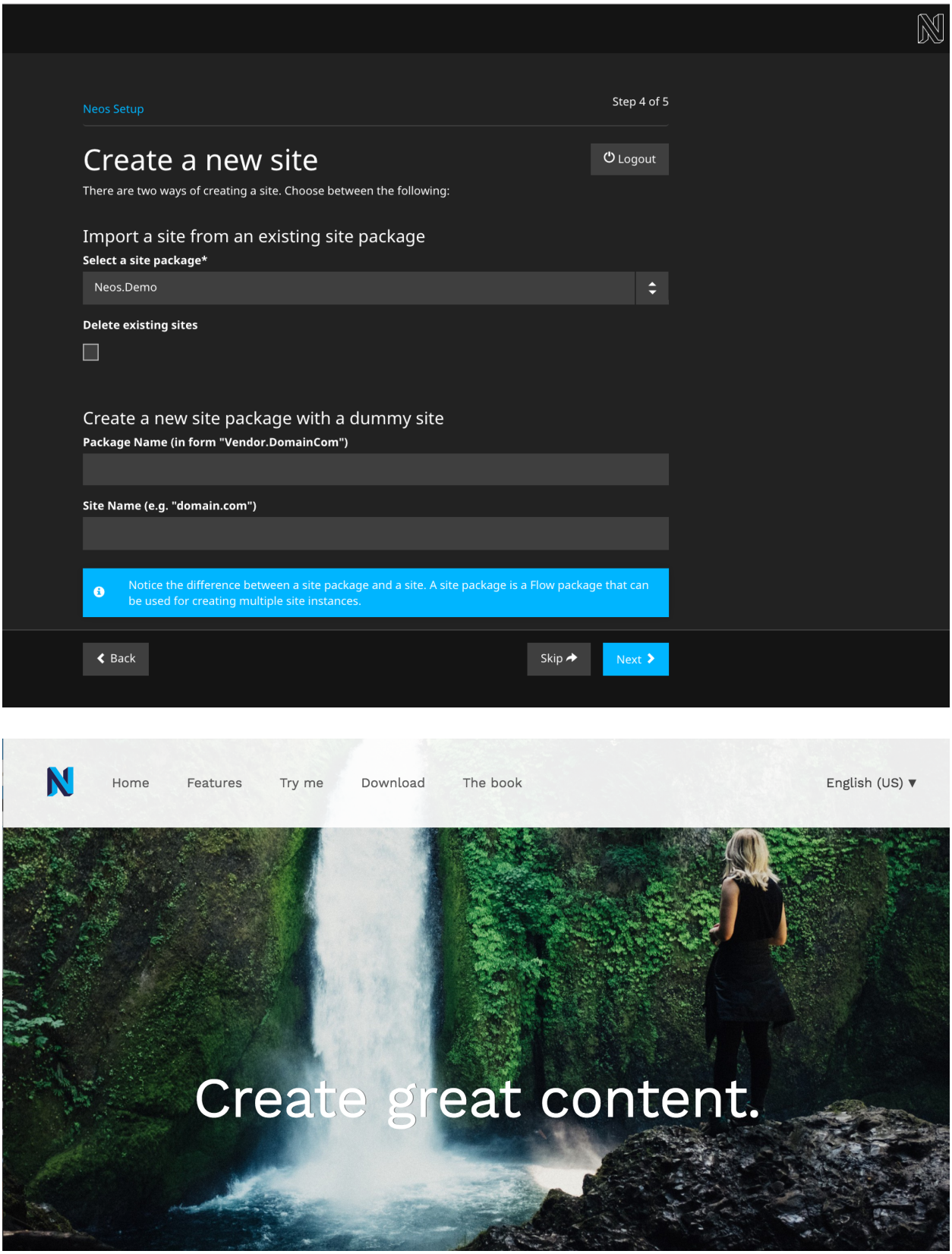


Fig. 1: The Neos demo site start page

CHAPTER

TWO

TECHNICAL PRINCIPLES

CHAPTER

THREE

CREATING A SITE WITH NEOS

This guide explains how to implement websites with Neos. It specifically covers the structuring of content using the *Neos Content Repository*, and how the content is rendered using *Fusion* and *Fluid*.

3.1 Node Types

These are the development guidelines of Neos.

3.1.1 Content Structure

Before we can understand how content is rendered, we have to see how it is structured and organized. These basics are explained in this section.

Nodes inside the Neos Content Repository

The content in Neos is not stored inside tables of a relational database, but inside a *tree-based* structure: the so-called Neos Content Repository.

To a certain extent, it is comparable to files in a file-system: They are also structured as a tree, and are identified uniquely by the complete path towards the file.

Note: Internally, the Neos ContentRepository currently stores the nodes inside database tables as well, but you do not need to worry about that as you'll never deal with the database directly. This high-level abstraction helps to decouple the data modelling layer from the data persistence layer.

Each element in this tree is called a *Node*, and is structured as follows:

- It has a *node name* which identifies the node, in the same way as a file or folder name identifies an element in your local file system.
- It has a *node type* which determines which properties a node has. Think of it as the type of a file in your file system.
- Furthermore, it has *properties* which store the actual data of the node. The *node type* determines which properties exist for a node. As an example, a `Text` node might have a `headline` and a `text` property.
- Of course, nodes may have *sub nodes* underneath them.

If we imagine a classical website with a hierarchical menu structure, then each of the pages is represented by a Neos ContentRepository Node of type `Document`. However, not only the pages themselves are represented as tree: Imagine a page has two columns, with different content elements inside each of them. The columns are stored as Nodes of type

`ContentCollection`, and they contain nodes of type `Text`, `Image`, or whatever structure is needed. This nesting can be done indefinitely: Inside a `ContentCollection`, there could be another three-column element which again contains `ContentCollection` elements with arbitrary content inside.

Predefined Node Types

Neos is shipped with a number of predefined node types. It is helpful to know some of them, as they can be useful elements to extend, and Neos depends on some of them for proper behavior.

There are a few core node types which are needed by Neos; these are shipped in `Neos.Neos` directly. All other node types such as Text, Image, ... are shipped inside the `Neos.NodeTypes` package.

Neos.Neos:Node

`Neos.Neos:Node` is a (more or less internal) base type which should be extended by all content types which are used in the context of Neos.

It does not define any properties.

Neos.Neos:Document

An important distinction is between nodes which look and behave like pages and "normal content" such as text, which is rendered inside a page. Nodes which behave like pages are called *Document Nodes* in Neos. This means they have a unique, externally visible URL by which they can be rendered.

The standard *page* in Neos is implemented by `Neos.NodeTypes:Page` which directly extends from `Neos.Neos:Document`.

Neos.Neos:ContentCollection and Neos.Neos:Content

All content which does not behave like pages, but which lives inside them, is implemented by two different node types:

First, there is the `Neos.Neos:ContentCollection` type: A `Neos.Neos:ContentCollection` has a structural purpose. It usually contains an ordered list of child nodes which are rendered inside.

`Neos.Neos:ContentCollection` may be extended by custom types.

Second, the node type for all standard elements (such as text, image, youtube, ...) is `Neos.Neos:Content`. This is–by far–the most often extended node type.

Extending the NodeTypes

To extend the existing NodeTypes or to create new ones please read at the *Node Type Definition* reference.

3.1.2 Node Type Definition

Each Neos ContentRepository Node (we'll just call it Node in the remaining text) has a specific *node type*. Node Types can be defined in any package by declaring them in `Configuration/NodeTypes.yaml`. If you have a rather large list of Node Types, you can also split your NodeType definitions into multiple `Configuration/NodeTypes.*.yaml` files for organizing them.

Each node type can have *one or multiple parent types*. If these are specified, all properties and settings of the parent types are inherited.

A node type definition can look as follows:

```
'My.Package:SpecialHeadline':
  superTypes:
    'Neos.Neos:Content': true
  ui:
    label: 'Special Headline'
    group: 'general'
  properties:
    headline:
      type: 'string'
      defaultValue: 'My Headline Default'
      ui:
        inlineEditable: true
      validation:
        'Neos.Neos/Validation/StringLengthValidator':
          minimum: 1
          maximum: 255
```

The following options are allowed:

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.

Legacy: In Neos versions before 4.0 it was required to use icons from the Fontawesome 3 or 4 versions, prefixed with "icon-"

Current: In Neos 4.0, Fontawesome 5 was introduced, enabling the usage of all free Fontawesome icons: https://fontawesome.com/icons?d=gallery&m=free Those can still 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".

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 *Translate NodeTypes*).

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.

`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 *Translate NodeTypes*).

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.

aloha Legacy configuration of rich text editor, works for the sake of backwards compatibility, but it is advised to use *inline.editorOptions* instead.

`inline`

editor A way to override default inline editor loaded for this property. Two editors are available out of the box: *ckeditor* (loads CKeditor4) and *ckeditor5* (loads CKeditor5). The default editor is configurable in Settings.yaml under the key *Neos.Neos.Ui.frontendConfiguration.defaultInlineEditor*. It is strongly recommended to start using CKeditor5 today, as the CKeditor4 integration will be deprecated and removed in the future versions. Additional custom inline editors are registered via the *inlineEditors* registry. See *Neos User Interface Extensibility API* for the detailed information on the topic.

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

Note: When using *inline.editorOptions* anything defined under the legacy *aloha* key for a property is ignored. Keep this in mind when using supertypes and mixins.

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
      u: true
      sub: true
      sup: true
      del: 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 Allows to observe changes of other properties in order to react to them. For details see *Depending Properties*

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'
```

(continues on next page)

(continued from previous page)

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

3.1.3 Node Type Constraints

In a typical Neos project, you will create lots of custom node types. However, many node types should only be used in a specific context and not everywhere. Neos allows you to define node type constraints, which restrict the possible node types that can be added as children of a specific node type. There are two ways to do this:

- Regular node type constraints are defined per node type. They apply in any context the node type appears in.
- Additionally, when a node type has auto-created child nodes (see *Node Type Definition*), you can define additional constraints that only apply for these child nodes. This allows you to restrict node type usage depending on the context that the node types are placed in.

Note: Node type constraints are cached in the browser's session storage. During development, it's a good idea to run `sessionStorage.clear()` in the browser console to remove the old configuration after you make changes. Alternatively, you can use an anonymous browser tab to avoid storing outdated node type constraints.

Regular Node Type Constraints

Let's assume that, inside the "Chapter" node type of the Neos Demo Site (which is a document node), one should only be able to create nested chapters, and not pages or shortcuts. Using node type constraints, this can be enforced:

```
'Neos.Demo:Chapter':
  constraints:
    nodeTypes:
      'Neos.Neos:Document': false
      'Neos.Demo:Chapter': true
```

In the above example, we disable all document node types using `'Neos.Neos:Document': false`, and then enable the `Neos.Demo:Chapter` node type as well as any node type that inherits from it. The reason why we use `'Neos.Neos:Document': false` instead of `'*': false` here is that by default, only document node types are allowed as children of other document node types anyway (see further down for more information regarding the defaults).

You might now wonder why it is still possible to create content inside the chapter (because everything except Chapter is disabled with the above configuration): The reason is that node type constraints are only enforced for nodes which are *not auto-created*. Because `Neos.Demo:Chapter` has an auto-created `main ContentCollection`, it is still possible to add content inside. In the following example, we see the node type definition which is shipped with the demo website:

```
'Neos.Demo:Chapter':
  superTypes:
    'Neos.Neos:Document': true
  childNodes:
    'main':
      type: 'Neos.Neos:ContentCollection'
```

The `main` ContentCollection is still added, even though you cannot add any more because ContentCollections are not allowed according to the node type constraints.

Auto-Created Child Node Constraints

Let's assume that our chapter node type should only contain text within its main ContentCollection. This is possible using additional constraints for each *auto-created child node*. These constraints will only be applied for the configured auto-created child nodes - not for any others, even if they are of the same type.

```
'Neos.Demo:Chapter':
  childNodes:
    'main':
      type: 'Neos.Neos:ContentCollection'
      constraints:
        nodeTypes:
          '*': false
          'Neos.NodeTypes:Text': true
```

Override Logic and Default Values

The following logic applies for node type constraints:

- Constraints are only enforced for child nodes which are not auto-created.
- You can specify node types explicitly or use '*' to allow/deny all node types.
- Setting the value to *true* is an explicit *allow*
- Setting the value to *false* is an explicit *deny*
- The default is to *always deny* (in case '*' is not specified).
- More specific constraints override less specific constraints. Specificity is deduced from the inheritance hierarchy of the node types. This means that e.g. setting *'*': false* will only apply if no more specific constraint has been set, such as *'Neos.Neos:Document': true*.
- Node type constraints are inherited from parent node types. If your node type has listed *Neos.Neos:Document* as a superType, its constraints will apply for your node type as well.

The last rule is especially important, since most node types you define will have either `Neos.NodeTypes:Page` (which, in turn, inherits from `Neos.Neos:Document`) or ``Neos.Neos:Content`` as superTypes. You should know which constraints are defined per default in order to effectively override them. These are the current defaults for these two node types - this is taken from `NodeTypes.yaml` in the Neos.Neos package.

```
'Neos.Neos:Document':
  constraints:
    nodeTypes:
      '*': false
      'Neos.Neos:Document': true
```

The document node type, by default, allows any other document node type below it. This means that if you want to disable all document node types under your custom one, setting `'*': false` will have no effect on anything inheriting from `Neos.Neos:Document` - the more specific constraint `'Neos.Neos:Document': true` will override it. You will need to set `'Neos.Neos:Document': false` instead.

The default content node type, on the other hand, only has the catch-all constraint. If you want to enable any child nodes, you can simply allow them.

```
'Neos.Neos:Content':
  constraints:
    nodeTypes:
      '*': false
```

Examples

You can use YAML references (with the `&xyz` and `*xyz` syntax) to re-use constraints. Here's how to disallow nested Two/Three/FourColumn inside a multi column element:

```
'Neos.NodeTypes:Column':
  childNodes:
    column0:
      constraints: &columnConstraints
        nodeTypes:
          'Neos.NodeTypes:TwoColumn': false
          'Neos.NodeTypes:ThreeColumn': false
          'Neos.NodeTypes:FourColumn': false
    column1:
      constraints: *columnConstraints
    column2:
      constraints: *columnConstraints
    column3:
      constraints: *columnConstraints
```

3.1.4 Node Creation Dialog Configuration

When creating new nodes, you have the possibility to provide additional data that will be passed to `nodeCreationHandlers`.

Creation dialog supports most of the inspector editors, except of those that require to show a secondary inspector view. See *Property Editor Reference* for more details about configuring inspector editors.

For example, this functionality is used in Neos to ask users for title before creating document nodes:

```
'Neos.Neos:Document':
  ui:
    group: 'general'
    creationDialog:
      elements:
        title:
          type: string
          ui:
            label: i18n
            editor: 'Neos.Neos/Inspector/Editors/TextFieldEditor'
          validation:
            'Neos.Neos/Validation/NotEmptyValidator': []
  options:
    nodeCreationHandlers:
      documentTitle:
        nodeCreationHandler: 'Neos\Neos\Ui\NodeCreationHandler\
↪DocumentTitleNodeCreationHandler'
```

You may register multiple `nodeCreationHandlers` per nodetype. Each nodeCreationHandler must implement `NodeCreationHandlerInterface`. It gets the newly created `$node` and the `$data` coming from the creation dialog.

Note: elements of the creation dialog define an arbitrary set of data that will be passed to a nodeCreationHandler, they will not automatically set node properties in any way. To take action based on that data you would need to write a custom node creation handler or use a package that already provides such functionality, e.g. Flowpack.NodeTemplates

(https://github.com/Flowpack/Flowpack.NodeTemplates).

3.1.5 Translate NodeTypes

To use the translations for NodeType labels or help messages you have to enable it for each label or message by setting the value to the predefined value "i18n".

NodeTypes.yaml

```
Vendor.Site:YourContentElementName:
  ui:
    help:
      message: 'i18n'
    inspector:
      tabs:
        yourTab:
          label: 'i18n'
      groups:
        yourGroup:
          label: 'i18n'
  properties:
    yourProperty:
      type: string
        ui:
          label: 'i18n'
          help:
            message: 'i18n'
```

That will instruct Neos to look for translations of these labels. To register an xliff file for this NodeTypes you have to add the following configuration to the Settings.yaml of your package:

```
Neos:
  Neos:
    userInterface:
      translation:
        autoInclude:
          'Vendor.Site': ['NodeTypes/*']
```

Inside of the xliff file **Resources/Private/Translations/en/NodeTypes/YourContentElementName.xlf** the translated labels for `help`, `properties`, `groups`, `tabs` and `views` are defined in the xliff as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
        <file original="" product-name="Vendor.Site" source-language="en" datatype=
↪"plaintext">
                <body>
                        <trans-unit id="ui.help.message" xml:space="preserve">
                                <source>Your help message here</source>
                        </trans-unit>
                        <trans-unit id="tabs.myTab" xml:space="preserve">
                                <source>Your Tab Title</source>
                        </trans-unit>
                        <trans-unit id="groups.myTab" xml:space="preserve">
                                <source>Your Group Title</source>
                        </trans-unit>
                        <trans-unit id="properties.myProperty" xml:space="preserve">
```

(continues on next page)

(continued from previous page)

```
                        <source>Your Property Title</source>
                    </trans-unit>
                    <trans-unit id="properties.myProperty.ui.help.message"␣
↪xml:space="preserve">
                        <source>Your help message here</source>
                    </trans-unit>
                </body>
        </file>
</xliff>
```

Add properties to existing NodeTypes

For adding properties to existing NodeTypes the use of mixins is encouraged.

NodeTypes.yaml

```
Vendor.Site:YourNodetypeMixin:
  abstract: true
  properties:
    demoProperty:
      type: string
        ui:
          label: 'i18n'

Neos.Neos:Page:
  superTypes:
    'Vendor.Site:YourNodetypeMixin': true
```

That way you can add the translations for the added properties to the file **Resources/Private/Translations/en/NodeTypes/YourNodetypeMixin.xlf**.

Override Translations

To override translations entirely or to use custom id's the label property can also contain a path of the format `Vendor.Package:Xliff.Path.And.Filename:labelType.identifier`. The string consists of three parts delimited by `:`:

- First, the *Package Key*
- Second, the path towards the xliff file, replacing slashes by dots (relative to `Resources/Private/Translations/<language>`).
- Third, the key inside the xliff file.

For the example above that would be `Vendor.Site:NodeTypes.YourContentElementName:properties.title`:

```
properties:
  title:
    type: string
      ui:
        label: 'Vendor.Site:NodeTypes.YourContentElementName:properties.title'
```

If you e.g. want to *relabel* an existing node property of a different package (like the `Neos.NodeTypes:Page`), you always have to specify the full translation key (pointing to your package's XLIFF files then).

Validate Translations

To validate that all labels are translated Neos has the following setting in *Settings.yaml*:

```
.. code-block:: yaml
```

Neos:

Neos:

userInterface: scrambleTranslatedLabels: true

If that setting is enabled all already translated labels are replaced with ###### – that way you can easily identify the labels that still lack translations.

Note: Make sure to flush the browser caches after working with the translation to make sure that the browser always shows the latest translations.

3.1.6 Dynamic Client-side Configuration Processing

Note: This API is rather low-level and still experimental, we might change some of the implementation details or compliment it with a more high-level API.

All configuration values that begin with `ClientEval:` are dynamically evaluated on the client side. They are written in plain JavaScript (evaluated with `eval`) and have `node` variable in the scope pointing to the currently focused node, with all transient inspector changes applied. For now it is only related to the nodetypes inspector configuration, but in the future may be extended to the other parts of the user interface.

A few Practical Examples

Hiding one property when the other one is not set

Here is an example how to hide the property `borderColor` if `borderWidth` is empty by changing its group name to a non-existant value:

```
'Some.Package:NodeType':
  properties:
    borderWidth:
      type: integer
      ui:
        inspector:
          group: 'style'
    borderColor:
      type: string
      ui:
        inspector:
          hidden: 'ClientEval:node.properties.borderWidth ? true : false
```

Dependent SelectBoxes

If you are using select box editors with *data sources* (see *Data sources* for more details) you can use client-side processing to adjust `dataSourceAdditionalData` when properties are changed in the inspector. The following example demonstrates this. It defines two properties (*serviceType* and *contractType*) where changes to the first property cause the `searchTerm` on the second properties' data source to be updated. That in turn triggers a refresh of the available options from the data source.

```
properties:
  serviceType:
    type: string
    ui:
      label: 'Service Type'
      inspector:
        group: product
        editor: 'Content/Inspector/Editors/SelectBoxEditor'
        editorOptions:
          allowEmpty: true
          placeholder: 'Service Type'
          dataSourceIdentifier: 'acme-servicetypes'
  contractType:
    type: string
    ui:
      label: 'Contract Type'
      inspector:
        group: product
        editor: 'Content/Inspector/Editors/SelectBoxEditor'
        editorOptions:
          allowEmpty: true
          placeholder: 'Contract Type'
          dataSourceIdentifier: 'acme-contracttypes'
          dataSourceAdditionalData:
            searchTerm: 'ClientEval:node.properties.serviceType'
```

3.1.7 Depending Properties

Note: This API is outdated and works only in the legacy (Ember) version of the Neos User Interface. For a relevant version of the API see *Dynamic Client-side Configuration Processing*.

Note: This API is still experimental, we might change details about the handler signature and implementation to reduce the amount of exposed internal code. The UI code is undergoing major changes right now which also might make adjustments necessary.

Sometimes it might be necessary to depend one property editor on another, such as two select boxes where one selection is not meaningful without the other. For that you can setup listeners that get triggered each time a property changes.

Here is an example of the configuration:

```
'Some.Package:NodeType':
  properties:
    border-width:
```

(continues on next page)

(continued from previous page)

```
      type: integer
    border-color:
      type: string
      ui:
        label: i18n
        inspector:
          editorListeners:
            activeWithNonEmptyValue:
              property: 'border-width'
              handler: 'Some.Package/Handlers/BorderHandler'
              handlerOptions:
                something: true
```

This sets up a listener named `activeWithNonEmptyValue`. The name can be freely chosen. This allows to override specific listeners in other packages by refering to that name. The `property` setting defines the name of the property on the same Node that will be observed. That means any change to this property will trigger the configured `handler`.

Configuring the `handler` means defining a require path to the handler object just like with *Custom Editors* for properties. Namespaces can be registered like this:

```
Neos:
  Neos:
    userInterface:
      requireJsPathMapping:
        'Some.Package/Handlers': 'resource://Some.Package/Public/Scripts/Inspector/
↪Handlers'
```

The handler should be compatible to RequireJS and be an `Ember.Object` that has a `handle` function. The `handlerOptions` configured for the listener in the NodeType configuration will be given to the handler upon creation and are available in the `handle` method.

A code example for a handler:

```
define(
[
    'emberjs'
],
function (Ember) {
    return Ember.Object.extend({
        handle: function(listeningEditor, newValue, property, listenerName) {
            if (this.get('something') === true) {
                listeningEditor.set('disabled', (newValue === null || newValue === '
↪'));
            }
        }
    });
});
```

The handle function receives the following arguments:

- `listeningEditor` - The property editor this listener is configured for, in the above example it will be the `border-color` editor.
- `newValue` will be the value of the observed property, which is the `border-width` probpery in the above example.
- `property` is the name of the observed property, literally `border-width` in the above example.

- `listenerName` is the configured name of the listener in question, literally `activeWithNonEmptyValue` in the example above.

If you are using select box editors with *data sources* (see *Data sources* for more details) you can use editor listeners to adjust `dataSourceAdditionalData` when properties are changed in the inspector. The following example shows this. It defines two properties (*serviceType* and *contractType*) where changes to the first property cause the `searchTerm` on the second properties' data source to be updated. That in turn triggers a refresh of the available options from the data source.

```
properties:
  serviceType:
    type: string
    ui:
      label: 'Service Type'
      inspector:
        group: product
        editor: 'Content/Inspector/Editors/SelectBoxEditor'
        editorOptions:
          allowEmpty: true
          placeholder: 'Service Type'
          dataSourceIdentifier: 'acme-servicetypes'
  contractType:
    type: string
    ui:
      label: 'Contract Type'
      inspector:
        group: product
        editor: 'Content/Inspector/Editors/SelectBoxEditor'
        editorOptions:
          allowEmpty: true
          placeholder: 'Contract Type'
          dataSourceIdentifier: 'acme-contracttypes'
          dataSourceAdditionalData:
            searchTerm: ~
        editorListeners:
          updateForSourceData:
            property: 'serviceType'
            handler: 'Neos.Demo/Handlers/TeaserOptionsHandler'
```

```
define(['emberjs'], function (Ember) {
  return Ember.Object.extend({
    handle: function(listeningEditor, newValue, property, listenerName) {
      listeningEditor.set('dataSourceAdditionalData.searchTerm', newValue);
    }
  });
});
```

3.1.8 Disable NodeTypes

To hide an existing NodeType (e.g. one that comes with Neos already) you have 2 options.

Hide the NodeType from the user interface

NodeTypes.yaml

```
'Vendor.Site:YourContentElementName':
  ui: ~
```

Nodes of this type will still remain valid in the database and being rendered to the frontend. But they will not be shown anymore in the dialog for adding nodes.

Completely disallow the direct usage of a NodeType

NodeTypes.yaml

```
'Vendor.Site:YourContentElementName':
  abstract: TRUE
```

As abstract NodeTypes are not valid to be used directly this will hide the NodeType in the user interface AND additionally make all existing nodes of this type invalid. If you run a *node:repair* all existing nodes of this type will be removed.

Note: Do not delete the complete NodeType via ~ because this will break all NodeTypes that inherit from this one.

3.2 Fusion

3.2.1 Inside Fusion

In this chapter, Fusion will be explained in a step-by-step fashion, focusing on the different internal parts, the syntax of these and the semantics.

Fusion is fundamentally a *hierarchical, prototype based processing language*:

- It is *hierarchical* because the content it should render is also hierarchically structured.
- It is *prototype based* because it allows to define properties for *all instances* of a certain Fusion object type. It is also possible to define properties not for all instances, but only for *instances inside a certain hierarchy*. Thus, the prototype definitions are hierarchically-scoped as well.
- It is a *processing language* because it processes the values in the *context* into a *single output value*.

In the first part of this chapter, the syntactic and semantic features of the Fusion, Eel and FlowQuery languages are explained. Then, the focus will be on the design decisions and goals of Fusion, to provide a better understanding of the main objectives while designing the language.

Goals of Fusion

Fusion should **cater to both planned and unplanned extensibility**. This means it should provide ways to adjust and extend its behavior in places where this is to be expected. At the same time it should also be possible to adjust and extend in any other place without having to apply dirty hacks.

Fusion should be **usable in standalone, extensible applications** outside of Neos. The use of a flexible language for configuration of (rendering) behavior is beneficial for most complex applications.

Fusion should make **out-of-band rendering** easy to do. This should ease content generation for technologies like AJAX or edge-side includes (ESI).

Fusion should make **multiple renderings of the same content** possible. It should allow placement of the same content (but possibly in different representations) on the same page multiple times.

Fusion's **syntax should be familiar to the user**, so that existing knowledge can be leveraged. To achieve this, Fusion takes inspiration from CSS selectors, jQuery and other technologies that are in widespread use in modern frontend development.

Fusion files

Fusion is read from files. In the context of Neos, some of these files are loaded automatically, and Fusion files can be split into parts to organize things as needed.

Automatic Fusion file inclusion

All Fusion files are expected to be in the package subfolder *Resources/Private/Fusion*. Neos will automatically include the file *Root.fusion* for the current site package (package which resides in Packages/Sites and has the type "neos-site" in its composer manifest).

To automatically include *Root.fusion* files from other packages, you will need to add those packages to the configuration setting `Neos.Neos.fusion.autoInclude`:

```
# Settings.yaml

Neos:
  Neos:
    fusion:
      autoInclude:
        Your.Package: true
```

Neos will then autoinclude *Root.fusion* files from these packages in the order defined by package management. Files with a name other than *Root.fusion* **will never be auto-included** even with that setting. You will need to include them manually in your *Root.fusion*.

Manual Fusion file inclusion

In any Fusion file further files can be included using the `include` statement. The path is either relative to the current file or can be given with the `resource` wrapper:

```
include: NodeTypes/CustomElements.fusion
include: resource://Acme.Demo/Private/Fusion/Quux.fusion
```

In addition to giving exact filenames, globbing is possible in two variants:

```
# Include all .fusion files in NodeTypes
include: NodeTypes/*

# Include all .fusion files in NodeTypes and it's subfolders recursively
include: NodeTypes/**/*
```

The first includes all Fusion files in the *NodeTypes* folder, the latter will recursively include all Fusion files in NodeTypes and any folders below.

The globbing can be combined with the `resource` wrapper:

```
include: resource://Acme.Demo/Private/Fusion/NodeTypes/*
include: resource://Acme.Demo/Private/Fusion/**/*
```

Fusion Objects

Fusion is a language to describe *Fusion objects*. A Fusion object has some *properties* which are used to configure the object. Additionally, a Fusion object has access to a *context*, which is a list of variables. The goal of a Fusion object is to take the variables from the context, and transform them to the desired *output*, using its properties for configuration as needed.

Thus, Fusion objects take some *input* which is given through the context and the properties, and produce a single *output value*. Internally, they can modify the context, and trigger rendering of nested Fusion objects: This way, a big task (like rendering a whole web page) can be split into many smaller tasks (render a single image, render some text, ...): The results of the small tasks are then put together again, forming the final end result.

Fusion object nesting is a fundamental principle of Fusion. As Fusion objects call nested Fusion objects, the rendering process forms a *tree* of Fusion objects.

Fusion objects are implemented by a PHP class, which is instantiated at runtime. A single PHP class is the basis for many Fusion objects. We will highlight the exact connection between Fusion objects and their PHP implementations later.

A Fusion object can be instantiated by assigning it to a Fusion path, such as:

```
foo = Page
# or:
my.object = Text
# or:
my.image = Neos.Neos.ContentTypes:Image
```

The name of the to-be-instantiated Fusion prototype is listed without quotes.

By convention, Fusion paths (such as `my.object`) are written in `lowerCamelCase`, while Fusion prototypes (such as `Neos.Neos.ContentTypes:Image`) are written in `UpperCamelCase`.

It is possible to set *properties* on the newly created Fusion objects:

```
foo.myProperty1 = 'Some Property which Page can access'
my.object.myProperty1 = "Some other property"
my.image.width = ${q(node).property('foo')}
```

Property values that are strings have to be quoted (with either single or double quotes). A property can also be an *Eel expression* (which are explained in *Eel, FlowQuery and Fizzle*.)

To reduce typing overhead, curly braces can be used to "abbreviate" long Fusion paths:

```
my {
  image = Image
  image.width = 200

  object {
    myProperty1 = 'some property'
  }
}
```

Instantiating a Fusion object and setting properties on it in a single pass is also possible. All three examples mean exactly the same:

```
someImage = Image
someImage.foo = 'bar'

# Instantiate object, set property one after each other
someImage = Image
someImage {
  foo = 'bar'
}

# Instantiate an object and set properties directly
someImage = Image {
  foo = 'bar'
}
```

Fusion Objects are Side-Effect Free

When Fusion objects are rendered, they are allowed to modify the Fusion context (they can add or override variables); and can invoke other Fusion objects. After rendering, however, the parent Fusion object must make sure to clean up the context, so that it contains exactly the state it had before the rendering.

The API helps to enforce this, as the Fusion context is a *stack*: The only thing the developer of a Fusion object needs to make sure is that if he adds some variable to the stack, effectively creating a new stack frame, he needs to remove exactly this stack frame after rendering again.

This means that a Fusion object can only manipulate Fusion objects *below it*, but not following or preceding it.

In order to enforce this, Fusion objects are furthermore only allowed to communicate through the Fusion Context; and they are never allowed to be invoked directly: Instead, all invocations need to be done through the *Fusion Runtime*.

All these constraints make sure that a Fusion object is *side-effect free*, leading to an important benefit: If somebody knows the exact path towards a Fusion object together with its context, it can be rendered in a stand-alone manner, exactly as if it was embedded in a bigger element. This enables, for example, rendering parts of pages with different cache life- times, or the effective implementation of AJAX or ESI handlers reloading only parts of a website.

Fusion Prototypes

When a Fusion object is instantiated (i.e. when you type `someImage = Image`) the *Fusion Prototype* for this object is *copied* and is used as a basis for the new object. The prototype is defined using the following syntax:

```
prototype(MyImage) {
        width = '500px'
        height = '600px'
}
```

When the above prototype is instantiated, the instantiated object will have all the properties of the copied prototype. This is illustrated through the following example:

```
someImage = MyImage
# now, someImage will have a width of 500px and a height of 600px

someImage.width = '100px'
# now, we have overridden the height of "someImage" to be 100px.
```

Prototype- vs. class-based languages

There are generally two major "flavours" of object-oriented languages. Most languages (such as PHP, Ruby, Perl, Java, C++) are *class-based*, meaning that they explicitly distinguish between the place where behavior for a given object is defined (the "class") and the runtime representation which contains the data (the "instance").

Other languages such as JavaScript are prototype-based, meaning that there is no distinction between classes and instances: At object creation time, all properties and methods of the object's *prototype* (which roughly corresponds to a "class") are copied (or otherwise referenced) to the *instance*.

Fusion is a *prototype-based language* because it *copies* the Fusion Prototype to the instance when an object is evaluated.

Prototypes in Fusion are *mutable*, which means that they can easily be modified:

```
prototype(MyYouTube) {
        width = '100px'
        height = '500px'
}

# you can change the width/height
prototype(MyYouTube).width = '400px'
# or define new properties:
prototype(MyYouTube).showFullScreen = ${true}
```

Defining and instantiating a prototype from scratch is not the only way to define and instantiate them. You can also use an *existing Fusion prototype* as basis for a new one when needed. This can be done by *inheriting* from a Fusion prototype using the < operator:

```
prototype(MyImage) < prototype(Neos.Neos:Content)

# now, the MyImage prototype contains all properties of the Template
# prototype, and can be further customized.
```

This implements *prototype inheritance*, meaning that the "subclass" (`MyImage` in the example above) and the "parent class (`Content`) are still attached to each other: If a property is added to the parent class, this also applies to the subclass, as in the following example:

```
prototype(Neos.Neos:Content).fruit = 'apple'
prototype(Neos.Neos:Content).meal = 'dinner'

prototype(MyImage) < prototype(Neos.Neos:Content)
# now, MyImage also has the properties "fruit = apple" and "meal = dinner"

prototype(Neos.Neos:Content).fruit = 'Banana'
# because MyImage *extends* Content, MyImage.fruit equals 'Banana' as well.

prototype(MyImage).meal = 'breakfast'
prototype(Neos.Fusion:Content).meal = 'supper'
# because MyImage now has an *overridden* property "meal", the change of
# the parent class' property is not reflected in the MyImage class
```

Prototype inheritance can only be defined *globally*, i.e. with a statement of the following form:

```
prototype(Foo) < prototype(Bar)
```

It is not allowed to nest prototypes when defining prototype inheritance, so the following examples are **not valid Fusion** and will result in an exception:

```
prototype(Foo) < some.prototype(Bar)
other.prototype(Foo) < prototype(Bar)
prototype(Foo).prototype(Bar) < prototype(Baz)
```

While it would be theoretically possible to support this, we have chosen not to do so in order to reduce complexity and to keep the rendering process more understandable. We have not yet seen a Fusion example where a construct such as the above would be needed.

Hierarchical Fusion Prototypes

One way to flexibly adjust the rendering of a Fusion object is done through modifying its *Prototype* in certain parts of the rendering tree. This is possible because Fusion prototypes are *hierarchical*, meaning that `prototype(...)` can be part of any Fusion path in an assignment; even multiple times:

```
prototype(Foo).bar = 'baz'
prototype(Foo).some.thing = 'baz2'

some.path.prototype(Foo).some = 'baz2'

prototype(Foo).prototype(Bar).some = 'baz2'
prototype(Foo).left.prototype(Bar).some = 'baz2'
```

- `prototype(Foo).bar` is a simple, top-level prototype property assignment. It means: *For all objects of type Foo, set property bar*. The second example is another variant of this pattern, just with more nesting levels inside the property assignment.
- `some.path.prototype(Foo).some` is a prototype property assignment *inside some.path*. It means: *For all objects of type Foo which occur inside the Fusion path some.path, the property some is set.*
- `prototype(Foo).prototype(Bar).some` is a prototype property assignment *inside another prototype*. It means: *For all objects of type Bar which occur somewhere inside an object of type Foo, the property some is set.*
- This can both be combined, as in the last example inside `prototype(Foo).left.prototype(Bar).some`.

Internals of hierarchical prototypes

A Fusion object is side-effect free, which means that it can be rendered deterministically knowing only its *Fusion path* and the *context*. In order to make this work with hierarchical prototypes, we need to encode the types of all Fusion objects above the current one into the current path. This is done using angular brackets:

```
a1/a2<Foo>/a3/a4<Bar>
```

When this path is rendered, `a1/a2` is rendered as a Fusion object of type `Foo` – which is needed to apply the prototype inheritance rules correctly.

Those paths are rarely visible on the "outside" of the rendering process, but might at times appear in exception messages if rendering fails. For those cases it is helpful to know their semantics.

Bottom line: It is not important to know exactly how the a rendering Fusion object's *Fusion path* is constructed. Just pass it on, without modification to render a single element out of band.

Namespaces of Fusion objects

The benefits of namespacing apply just as well to Fusion objects as they apply to other languages. Namespacing helps to organize the code and avoid name clashes.

In Fusion the namespace of a prototype is given when the prototype is declared. The following declares a `YouTube` prototype in the `Acme.Demo` namespace:

```
prototype(Acme.Demo:YouTube) {
        width = '100px'
        height = '500px'
}
```

The namespace is, by convention, the package key of the package in which the Fusion resides.

Fully qualified identifiers can be used everywhere an identifier is used:

```
prototype(Neos.Neos:ContentCollection) < prototype(Neos.Neos:Collection)
```

In Fusion a `default` namespace of `Neos.Fusion` is set. So whenever `Value` is used in Fusion, it is a shortcut for `Neos.Fusion:Value`.

Custom namespace aliases can be defined using the following syntax:

```
namespace: Foo = Acme.Demo

# the following two lines are equivalent now
video = Acme.Demo:YouTube
video = Foo:YouTube
```

Warning: These declarations are scoped to the file they are in and have to be declared in every fusion file where they shall be used.

Setting Properties On a Fusion Object

Although the Fusion object can read its context directly, it is good practice to instead use *properties* for configuration:

```
# imagine there is a property "foo=bar" inside the Fusion context at this point
myObject = MyObject

# explicitly take the "foo" variable's value from the context and pass it into the
↪"foo"
# property of myObject. This way, the flow of data is more visible.
myObject.foo = ${foo}
```

While `myObject` could rely on the assumption that there is a `foo` variable inside the Fusion context, it has no way (besides written documentation) to communicate this to the outside world.

Therefore, a Fusion object's implementation should *only use properties* of itself to determine its output, and be independent of what is stored in the context.

However, in the prototype of a Fusion object it is perfectly legal to store the mapping between the context variables and Fusion properties, such as in the following example:

```
# this way, an explicit default mapping between a context variable and a property of␣
↪the
# Fusion object is created.
prototype(MyObject).foo = ${foo}
```

To sum it up: When implementing a Fusion object, it should not access its context variables directly, but instead use a property. In the Fusion object's prototype, a default mapping between a context variable and the prototype can be set up.

Default Context Variables

Neos exposes some default variables to the Fusion context that can be used to control page rendering in a more granular way.

- `node` can be used to get access to the current node in the node tree and read its properties. It is of type `NodeInterface` and can be used to work with node data, such as:

  ```
  # Make the node available in the template
  node = ${node}

  # Expose the "backgroundImage" property to the rendering using FlowQuery
  backgroundImage = ${q(node).property('backgroundImage')}
  ```

  To see what data is available on the node, you can expose it to the template as above and wrap it in a debug view helper:

  ```
  {node -> f:debug()}
  ```

- `documentNode` contains the closest parent document node - broadly speaking, it is the page the current node is on. Just like `node`, it is a `NodeInterface` and can be provided to the rendering in the same way:

  ```
  # Expose the document node to the template
  documentNode = ${documentNode}

  # Display the document node path
  nodePath = ${documentNode.path}
  ```

`documentNode` is in the end just a shorthand to get the current document node faster. It could be replaced with:

```
# Expose the document node to the template using FlowQuery and a Fizzle operator
documentNode = ${q(node).closest('[instanceof Neos.Neos:Document]').get(0)}
```

- `request` is an instance of `Neos\Flow\Mvc\ActionRequest` and allows you to access the current request from within Fusion. Use it to provide request variables to the template:

```
# This would provide the value sent by an input field with name="username".
userName = ${request.arguments.username}

# request.format contains the format string of the request, such as "html" or
↪"json"
requestFormat = ${request.format}
```

Another use case is to trigger an action, e.g. a search, via a custom Eel helper:

```
searchResults = ${Search.query(site).fulltext(request.arguments.searchword).
↪execute()}
```

A word of caution: You should never trigger write operations from Fusion, since it can be called multiple times (or not at all, because of caching) during a single page render. If you want a request to trigger a persistent change on your site, it's better to use a Plugin.

Manipulating the Fusion Context

The Fusion context can be manipulated directly through the use of the `@context` meta-property:

```
myObject = MyObject
myObject.@context.bar = ${foo * 2}
```

In the above example, there is now an additional context variable `bar` with twice the value of `foo`.

This functionality is especially helpful if there are strong conventions regarding the Fusion context variables. This is often the case in standalone Fusion applications, but for Neos, this functionality is hardly ever used.

Warning: In order to prevent unwanted side effects, it is not possible to access context variables from within `@context` on the same level. This means that the following will never return the string `Hello World!`

@context.contextOne = 'World!' @context.contextTwo = ${'Hello ' + contextOne} output = ${contextTwo}

Processors

Processors allow the manipulation of values in Fusion properties. A processor is applied to a property using the `@process` meta-property:

```
myObject = MyObject {
        property = 'some value'
        property.@process.1 = ${'before ' + value + ' after'}
}
# results in 'before some value after'
```

Multiple processors can be used, their execution order is defined by the numeric position given in the Fusion after `@process`. In the example above a `@process.2` would run on the results of `@process.1`.

Additionally, an extended syntax can be used as well:

```
myObject = MyObject {
        property = 'some value'
        property.@process.someWrap {
                expression = ${'before ' + value + ' after'}
                @position = 'start'
        }
}
```

This allows to use string keys for the processor name, and support `@position` arguments as explained for Arrays.

Processors are Eel Expressions or Fusion objects operating on the `value` property of the context. Additionally, they can access the current Fusion object they are operating on as `this`.

Conditions

Conditions can be added to all values to prevent evaluation of the value. A condition is applied to a property using the `@if` meta-property:

```
myObject = Menu {
        @if.1 = ${q(node).property('showMenu') == true}
}
# results in the menu object only being evaluated if the node's showMenu property is␣
↪not ``false``
# the php rules for mapping values to boolean are used internally so following values␣
↪are
# considered beeing false: ``null, false, '', 0, []``
```

Multiple conditions can be used, and if one of them doesn't return `true` the condition stops evaluation.

Apply

`@apply` allows to override multiple properties of a fusion-prototype with a single expression. This is useful when complex data structures are mapped to fusion prototypes.

The example shows the rendering of a `teaserList`-array by using a Teaser-Component and passing all keys from each teaser to the fusion Object:

```
teasers = Neos.Fusion:Collection {
        collection = ${teaserList}
        itemName = 'teaser'
        itemRenderer = Vendor.Site:Teaser {
                @apply.teaser = ${teaser}
        }
}
```

The code avoids passing down each fusion-property explicitly to the child component. A similar concept with different syntax from the JavaScript world is known as ES6-Spreads.

Another use-case is to use `Neos.Fusion:Renderer` to render a prototype while type and properties are based on data from the context:

```
example = Neos.Fusion:Renderer {
        type = ${data.type}
        element.@apply.properties = ${data.properties}
}
```

That way some meta-programming can used in fusion and both prototype and properties are decided late in the rendering by the fusion runtime.

How it works

The keys below `@apply` are evaluated before the fusion-object and the `@context` and are initialized. Each key below `@apply` must return a key-value map (values other than an array it are ignored). During the evaluation of each fusion-path the values from `@apply` are always checked first.

If a property is defined via `@apply` this value is returned without evaluating the fusionPath.

The `@process` and `@if`-rules of the original fusion-key are still applied even if a value from `@apply` is returned.

Since `@apply` is evaluated first the overwritten values are already present during the evaluation of `@context` and will overlay the properties of `this` if they are accessed.

`@apply` supports the same extended syntax and ordering as fusion processors and supports multiple keys. The evaluation order is defined via `@position`, the keys that are evaluated last will override previously defined keys. This is also similar to the rules for `@process`:

```
test = Vendor.Site:Prototype {
        @apply.contextValue {
                @position = 'start'
                expression = ${ arrayValueFromContext }
        }
        @apply.fusionObject {
                @position = 'end'
                expression = Neos.Fusion:RawArray {
                        value = "altered value"
                }
        }
}
```

Other than `@context` `@apply` is only valid for a single fusion path, so when subpathes or children are rendered they are not affected by the parents `@apply` unless they are explicitly passed down.

Debugging

To show the result of Fusion Expressions directly you can use the Neos.Fusion:Debug Fusion-Object:

```
debugObject = Neos.Fusion:Debug {
        # optional: set title for the debug output
        # title = 'Debug'

        # optional: show result as plaintext
        # plaintext = TRUE

        # If only the "value"-key is given it is debugged directly,
        # otherwise all keys except "title" and "plaintext" are debugged.
        value = "hello neos world"
```

(continues on next page)

(continued from previous page)

```
        # Additional values for debugging
        documentTitle = ${q(documentNode).property('title')}
        documentPath = ${documentNode.path}
}
# the value of this object is the formatted debug output of all keys given to the␣
↪object
```

Domain-specific languages in Fusion

Fusion allows the implementation of domain-specific sublanguages. Those DSLs can take a piece of code, that is optimized to express a specific class of problems, and return the equivalent fusion-code that is cached and executed by the Fusion-runtime afterwards.

Fusion-DSLs use the syntax of tagged template literals from ES6 and can be used in all value assignments:

```
value = dslIdentifier`... the code that is passed to the dsl ...`
```

If such a syntax-block is detected fusion will:

- Lookup the key `dslIdentifier` in the Setting `Neos.Fusion.dsl` to find the matching dsl-implementation.
- Instantiate the dsl-implementation class that was found registered.
- Check that the dsl-implementation satisfies the interface `\Neos\Fusion\Core\DslInterface`
- Pass the code between the backticks to the dsl-implementation.
- Finally parse the returned Fusion-code

Fusion DSLs cannot extend the fusion-language and -runtime itself, they are meant to enable a more efficient syntax for specific problems.

3.2.2 Eel, FlowQuery and Fizzle

Eel - Embedded Expression Language

Besides simple Fusion assignments such as `myObject.foo = 'bar'`, it is possible to write *expressions* using the *Eel* language such as `myObject.foo = ${q(node).property('bar')}`.

The *Embedded Expression Language* (Eel) is a building block for creating Domain Specific Languages. It provides a rich *syntax* for arbitrary expressions, such that the author of the DSL can focus on its Semantics.

In this section, the focus lies on the use of Eel inside Fusion.

Syntax

Every Eel expression in Fusion is surrounded by `${...}`, which is the delimiter for Eel expressions. Basically, the Eel syntax and semantics is like a condensed version of JavaScript:

- Most things you can write as a single JavaScript expression (that is, without a `;`) can also be written as Eel expression.
- Eel does not throw an error if `null` values are dereferenced, i.e. inside `${foo.bar}` with `foo` being `null`. Instead, `null` is returned. This also works for calling undefined functions.
- Eel does not support control structures or variable declarations.
- Eel supports the common JavaScript arithmetic and comparison operators, such as `+-*/%` for arithmetic and `== != > >= < <=` for comparison operators. Operator precedence is as expected, with multiplication binding higher than addition. This can be adjusted by using brackets. Boolean operators `&&` and `||` are supported.
- Eel supports the ternary operator to allow for conditions `<condition> ? <ifTrue> : <ifFalse>`.
- When object access is done (such as `foo.bar.baz`) on PHP objects, getters are called automatically.
- Object access with the offset notation is supported as well: `foo['bar']`

This means the following expressions are all valid Eel expressions:

```
${foo.bar}          // Traversal
${foo.bar()}        // Method call
${foo.bar().baz()} // Chained method call

${foo.bar("arg1", true, 42)} // Method call with arguments

${12 + 18.5}          // Calculations are possible
${foo == bar}       // ... and comparisons

${foo.bar(12+18.5, foo == bar)} // and of course also use it inside arguments

${[foo, bar]}            // Array Literal
${{foo: bar, baz: test}} // Object Literal
```

Semantics inside Fusion

Eel does not define any functions or variables by itself. Instead, it exposes the *Eel context array*, meaning that functions and objects which should be accessible can be defined there.

Because of that, Eel is perfectly usable as a "domain-specific language construction kit", which provides the syntax, but not the semantics of a given language.

For Eel inside Fusion, the semantics are as follows:

- All variables of the Fusion context are made available inside the Eel context.
- The special variable `this` always points to the current Fusion object implementation.
- The function `q()` is available, which wraps its argument into a FlowQuery object. *FlowQuery* is explained below.

By default the following Eel helpers are available in the default context for Eel expressions:

- `String`, exposing `Neos\Eel\Helper\StringHelper`
- `Array`, exposing `Neos\Eel\Helper\ArrayHelper`

- `Date`, exposing `Neos\Eel\Helper\DateHelper`
- `Configuration`, exposing `Neos\Eel\Helper\ConfigurationHelper`
- `Math`, exposing `Neos\Eel\Helper\MathHelper`
- `Json`, exposing `Neos\Eel\Helper\JsonHelper`
- `Security`, exposing `Neos\Eel\Helper\SecurityHelper`
- `Translation`, exposing `Neos\Flow\I18n\EelHelper\TranslationHelper`
- `Neos.Node`, exposing `Neos\Neos\Fusion\Helper\NodeHelper`
- `Neos.Link`, exposing `Neos\Neos\Fusion\Helper\LinkHelper`
- `Neos.Array`, exposing `Neos\Neos\Fusion\Helper\ArrayHelper`
- `Neos.Rendering`, exposing `Neos\Neos\Fusion\Helper\RenderingHelper`

See: *Eel Helpers Reference*

This is configured via the setting `Neos.Fusion.defaultContext`.

Additionally, the defaultContext contains the `request` object, where you have also access to Arguments. e.g. `${request.httpRequest.arguments.nameOfYourGetArgument}`

FlowQuery

FlowQuery, as the name might suggest, *is like jQuery for Flow*. It's syntax has been heavily influenced by jQuery.

FlowQuery is a way to process the content (being a Neos ContentRepository node within Neos) of the Eel context. FlowQuery operations are implemented in PHP classes. For any FlowQuery operation to be available, the package containing the operation must be installed. Any package can add their own FlowQuery operations. A set of basic operations is always available as part of the Neos.Eel package itself.

In Neos.Neos, the following FlowQuery operations are defined:

`property` Adjusted to access properties of a Neos ContentRepository node. If property names are prefixed with an underscore, internal node properties like start time, end time, and hidden are accessed.

`filter` Used to check a value against a given constraint. The filters expressions are given in *Fizzle*, a language inspired by CSS selectors. The Neos-specific filter changes `instanceof` to work on node types instead of PHP classes.

`children` Returns the children of a Neos ContentRepository node. They are optionally filtered with a `filter` operation to limit the returned result set.

`parents` Returns the parents of a Neos ContentRepository node. They are optionally filtered with a `filter` operation to limit the returned result set.

A reference of all FlowQuery operations defined in Neos.Eel and Neos.Neos can be found in the *FlowQuery Operation Reference*.

Operation Resolving

When multiple packages define an operation with the same short name, they are resolved using the priority each implementation defines, higher priorities have higher precedence when operations are resolved.

The `OperationResolver` loops over the implementations sorted by order and asks them if they can evaluate the current context. The first operation that answers this check positively is used.

FlowQuery by Example

Any context variable can be accessed directly:

```
${myContextVariable}
```

and the current node is available as well:

```
${node}
```

There are various ways to access its properties. Direct access is possible, but should be avoided. It is better to use FlowQuery instead:

```
${q(node).getProperty('foo')} // Possible, but discouraged
${q(node).property('foo')} // Better: use FlowQuery instead
```

Through this a node property can be fetched and assigned to a variable:

```
text = ${q(node).property('text')}
```

Fetching all parent nodes of the current node:

```
${q(node).parents()}
```

Here are two equivalent ways to fetch the first node below the `left` child node:

```
${q(node).children('left').first()}
${q(node).children().filter('left').first()}
```

Fetch all parent nodes and add the current node to the selected set:

```
${node.parents().add(node)}
```

The next example combines multiple operations. First it fetches all children of the current node that have the name `comments`. Then it fetches all children of those nodes that have a property `spam` with a value of false. The result of that is then passed to the `count()` method and the count of found nodes is assigned to the variable 'numberOfComments':

```
numberOfComments = ${q(node).children('comments').children("[spam = false]").count()}
```

The following expands a little more on that. It assigns a set of nodes to the `collection` property of the comments object. This set of nodes is either fetched from different places, depending on whether the current node is a `ContentCollection` node or not. If it is, the children of the current node are used directly. If not, the result of `this.getNodePath()` is used to fetch a node below the current node and those children are used. In both cases the nodes are again filtered by a check for their property `spam` being false.

```
comments.collection = ${q(node).is('[instanceof Neos.Neos:ContentCollection]') ?
        q(node).children("[spam = false]") : q(node).children(this.getNodePath()).
↪children("[spam = false]")}
```

Querying for nodes of two or more different node types

```
elements = ${q(node).filter('[instanceof Neos.NodeTypes:Text],[instanceof Neos.
↪NodeTypes:TextWithImage]').get()}
```

Fizzle

Filter operations as already shown are written in *Fizzle*. It has been inspired by the selector syntax known from CSS.

Property Name Filters

The first component of a filter query can be a `Property Name` filter. It is given as a simple string. Checks against property paths are not currently possible:

```
foo          //works
foo.bar      //does not work
foo.bar.baz  //does not work
```

In the context of Neos the property name is rarely used, as FlowQuery operates on Neos ContentRepository nodes and the `children` operation has a clear scope. If generic PHP objects are used, the property name filter is essential to define which property actually contains the `children`.

Attribute Filters

The next component are `Attribute` filters. They can check for the presence and against the values of attributes of context elements:

```
baz[foo]
baz[answer = 42]
baz[foo = "Bar"]
baz[foo = 'Bar']
baz[foo != "Bar"]
baz[foo ^= "Bar"]
baz[foo $= "Bar"]
baz[foo *= "Bar"]
```

As the above examples show, string values can be quoted using double or single quotes.

Available Operators

The operators for checking against attribute are as follows:

= Strict equality of value and operand

!= Strict inequality of value and operand

$= Value ends with operand (string-based)

^= Value starts with operand (string-based)

***=** Value contains operand (string-based)

instanceof Checks if the value is 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 class name with the PHP instanceof operation to check if the value matches.

Using Multiple Filters

It is possible to combine multiple filters:

[foo][bar][baz] All filters have to match (AND)

[foo],[bar],[baz] Only one filter has to match (OR)

3.3 Rendering Custom Markup

These are the development guidelines of Neos.

3.3.1 Templating

Templating is done in *Fluid*, which is a next-generation templating engine. It has several goals in mind:

- Simplicity
- Flexibility
- Extensibility
- Ease of use

This templating engine should not be bloated, instead, we try to do it "The Zen Way" - you do not need to learn too many things, thus you can concentrate on getting your things done, while the template engine handles everything you do not want to care about.

What Does it Do?

In many MVC systems, the view currently does not have a lot of functionality. The standard view usually provides a `render` method, and nothing more. That makes it cumbersome to write powerful views, as most designers will not write PHP code.

That is where the Template Engine comes into play: It "lives" inside the View, and is controlled by a special TemplateView which instantiates the Template Parser, resolves the template HTML file, and renders the template afterwards.

Below, you'll find a snippet of a real-world template displaying a list of blog postings. Use it to check whether you find the template language intuitive:

```
{namespace f=Neos\FluidAdaptor\ViewHelpers}
<html>
<head><title>Blog</title></head>
<body>
<h1>Blog Postings</h1>
<f:for each="{postings}" as="posting">
  <h2>{posting.title}</h2>
  <div class="author">{posting.author.name} {posting.author.email}</div>
  <p>
    <f:link.action action="details" arguments="{id : posting.id}">
      {posting.teaser}
    </f:link.action>
  </p>
</f:for>
</body>
</html>
```

- The *Namespace Import* makes the `\Neos\FluidAdaptor\ViewHelper` namespace available under the shorthand f.
- The `<f:for>` essentially corresponds to `foreach ($postings as $posting)` in PHP.
- With the dot-notation (`{posting.title}` or `{posting.author.name}`), you can traverse objects. In the latter example, the system calls `$posting->getAuthor()->getName()`.
- The `<f:link.action />` tag is a so-called ViewHelper. It calls arbitrary PHP code, and in this case renders a link to the "details"-Action.

There is a lot more to show, including:

- Layouts
- Custom View Helpers
- Boolean expression syntax

We invite you to explore Fluid some more, and please do not hesitate to give feedback!

Basic Concepts

This section describes all basic concepts available. This includes:

- Namespaces
- Variables / Object Accessors
- View Helpers
- Arrays

Namespaces

Fluid can be extended easily, thus it needs a way to tell where a certain tag is defined. This is done using namespaces, closely following the well-known XML behavior.

Namespaces can be defined in a template in two ways:

{namespace f=NeosFluidAdaptorViewHelpers} This is a non-standard way only understood by Fluid. It links the `f` prefix to the PHP namespace `\Neos\FluidAdaptor\ViewHelpers`.

<html xmlns:foo="http://some/unique/namespace"> The standard for declaring a namespace in XML. This will link the `foo` prefix to the URI `http://some/unique/namespace` and Fluid can look up the corresponding PHP namespace in your settings (so this is a two-piece configuration). This makes it possible for your XML editor to validate the template files and even use an XSD schema for auto completion.

A namespace linking `f` to `\Neos\FluidAdaptor\ViewHelpers` is imported by default. All other namespaces need to be imported explicitly.

If using the XML namespace syntax the default pattern `http://typo3.org/ns/<php namespace>` is resolved automatically by the Fluid parser. If you use a custom XML namespace URI you need to configure the URI to PHP namespace mapping. The YAML syntax for that is:

```
Neos:
  Fluid:
    namespaces:
      'http://some/unique/namespace': 'My\Php\Namespace'
```

Variables and Object Accessors

A templating system would be quite pointless if it was not possible to display some external data in the templates. That's what variables are for.

Suppose you want to output the title of your blog, you could write the following snippet into your controller:

```
$this->view->assign('blogTitle', $blog->getTitle());
```

Then, you could output the blog title in your template with the following snippet:

```
<h1>This blog is called {blogTitle}</h1>
```

Now, you might want to extend the output by the blog author as well. To do this, you could repeat the above steps, but that would be quite inconvenient and hard to read.

Note: The semantics between the controller and the view should be the following: The controller instructs the view to "render the blog object given to it", and not to "render the Blog title, and the blog posting 1, . . . ".

Passing objects to the view instead of simple values is highly encouraged!

That is why the template language has a special syntax for object access. A nicer way of expressing the above is the following:

```
// This should go into the controller:
$this->view->assign('blog', $blog);
```

```
<!-- This should go into the template: -->
<h1>This blog is called {blog.title}, written by {blog.author}</h1>
```

Instead of passing strings to the template, we are passing whole objects around now - which is much nicer to use both from the controller and the view side. To access certain properties of these objects, you can use Object Accessors. By writing `{blog.title}`, the template engine will call a `getTitle()` method on the blog object, if it exists. Besides, you can use that syntax to traverse associative arrays and public properties.

Tip: Deep nesting is supported: If you want to output the email address of the blog author, then you can use `{blog.author.email}`, which is roughly equivalent to `$blog->getAuthor()->getEmail()`.

View Helpers

All output logic is placed in View Helpers.

The view helpers are invoked by using XML tags in the template, and are implemented as PHP classes (more on that later).

This concept is best understood with an example:

```
{namespace f=Neos\FluidAdaptor\ViewHelpers}
<f:link.action controller="Administration">Administration</f:link.action>
```

The example consists of two parts:

- *Namespace Declaration* as explained earlier.
- *Calling the View Helper* with the `<f:link.action...> ... </f:link.action>` tag renders a link.

Now, the main difference between Fluid and other templating engines is how the view helpers are implemented: For each view helper, there exists a corresponding PHP class. Let's see how this works for the example above:

The `<f:link.action />` tag is implemented in the class `\Neos\FluidAdaptor\ViewHelpers\Link\ActionViewHelper`.

Note: The class name of such a view helper is constructed for a given tag as follows:

1. The first part of the class name is the namespace which was imported (the namespace prefix `f` was expanded to its full namespace `Neos\FluidAdaptor\ViewHelpers`)
2. The unqualified name of the tag, without the prefix, is capitalized (`Link`), and the postfix ViewHelper is appended.

The tag and view helper concept is the core concept of Fluid. All output logic is implemented through such ViewHelpers / tags! Things like `if/else`, `for`, `...` are all implemented using custom tags - a main difference to other templating languages.

Note: Some benefits of the class-based approach approach are:

- You cannot override already existing view helpers by accident.
- It is very easy to write custom view helpers, which live next to the standard view helpers
- All user documentation for a view helper can be automatically generated from the annotations and code documentation.

Most view helpers have some parameters. These can be plain strings, just like in `<f:link.action controller="Administration">...</f:link.action>`, but as well arbitrary objects. Parameters of view helpers will just be parsed with the same rules as the rest of the template, thus you can pass arrays or objects as parameters.

This is often used when adding arguments to links:

```
<f:link.action controller="Blog" action="show" arguments="{singleBlog: blogObject}">
  ... read more
</f:link.action>
```

Here, the view helper will get a parameter called `arguments` which is of type `array`.

Warning: Make sure you do not put a space before or after the opening or closing brackets of an array. If you type `arguments=" {singleBlog : blogObject}"` (notice the space before the opening curly bracket), the array is automatically casted to a string (as a string concatenation takes place).

This also applies when using object accessors: `<f:do.something with="{object}" />` and `<f:do.something with=" {object}" />` are substantially different: In the first case, the view helper will receive an object as argument, while in the second case, it will receive a string as argument.

This might first seem like a bug, but actually it is just consistent that it works that way.

Boolean Expressions

Often, you need some kind of conditions inside your template. For them, you will usually use the `<f:if>` ViewHelper. Now let's imagine we have a list of blog postings and want to display some additional information for the currently selected blog posting. We assume that the currently selected blog is available in `{currentBlogPosting}`. Now, let's have a look how this works:

```
<f:for each="{blogPosts}" as="post">
  <f:if condition="{post} == {currentBlogPosting}">... some special output here ...</
↪f:if>
</f:for>
```

In the above example, there is a bit of new syntax involved: `{post} == {currentBlogPosting}`. Intuitively, this says "if the post I"m currently iterating over is the same as currentBlogPosting, do something."

Why can we use this boolean expression syntax? Well, because the `IfViewHelper` has registered the argument condition as `boolean`. Thus, the boolean expression syntax is available in all arguments of ViewHelpers which are of type `boolean`.

All boolean expressions have the form `X <comparator> Y`, where:

- *<comparator>* is one of the following: `==`, `>`, `>=`, `<`, `<=`, `%` (modulo)
- *X* and *Y* are one of the following:

- a number (integer or float)
- a string (in single or double quotes)
- a JSON array
- a ViewHelper
- an Object Accessor (this is probably the most used example)
- inline notation for ViewHelpers

Inline Notation for ViewHelpers

In many cases, the tag-based syntax of ViewHelpers is really intuitive, especially when building loops, or forms. However, in other cases, using the tag-based syntax feels a bit awkward – this can be demonstrated best with the `<f:uri.resource>`- ViewHelper, which is used to reference static files inside the *Public/* folder of a package. That's why it is often used inside `<style>` or `<script>`-tags, leading to the following code:

```
<link rel="stylesheet" href="<f:uri.resource path='myCssFile.css' />" />
```

You will notice that this is really difficult to read, as two tags are nested into each other. That's where the inline notation comes into play: It allows the usage of `{f:uri.resource()}` instead of `<f:uri.resource />`. The above example can be written like the following:

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

This is readable much better, and explains the intent of the ViewHelper in a much better way: It is used like a helper function.

The syntax is still more flexible: In real-world templates, you will often find code like the following, formatting a `DateTime` object (stored in `{post.date}` in the example below):

```
<f:format.date format="d-m-Y">{post.date}</f:format.date>
```

This can also be re-written using the inline notation:

```
{post.date -> f:format.date(format:'d-m-Y')}
```

This is also a lot better readable than the above syntax.

Tip: This can also be chained indefinitely often, so one can write:

```
{post.date -> foo:myHelper() -> bar:bla()}
```

Sometimes you'll still need to further nest ViewHelpers, that is when the design of the ViewHelper does not allow that chaining or provides further arguments. Have in mind that each argument itself is evaluated as Fluid code, so the following constructs are also possible:

```
{foo: bar, baz: '{planet.manufacturer -> f:someother.helper(test: \'stuff\')}'}
{some: '{f:format.stuff(arg: \'foo'\)}'}
```

To wrap it up: Internally, both syntax variants are handled equally, and every ViewHelper can be called in both ways. However, if the ViewHelper "feels" like a tag, use the tag-based notation, if it "feels" like a helper function, use the Inline Notation.

Arrays

Some view helpers, like the `SelectViewHelper` (which renders an HTML select dropdown box), need to get associative arrays as arguments (mapping from internal to displayed name). See the following example for how this works:

```
<f:form.select options="{edit: 'Edit item', delete: 'Delete item'}" />
```

The array syntax used here is very similar to the JSON object syntax. Thus, the left side of the associative array is used as key without any parsing, and the right side can be either:

- a number:

```
{a : 1,
 b : 2
}
```

- a string; Needs to be in either single- or double quotes. In a double-quoted string, you need to escape the `"` with a `\` in front (and vice versa for single quoted strings). A string is again handled as Fluid Syntax, this is what you see in example `c`:

```
{a : 'Hallo',
 b : "Second string with escaped \" (double quotes) but not escaped ' (single␣
↪quotes)"
 c : "{firstName} {lastName}"
}
```

- a boolean, best represented with their integer equivalents:

```
{a : 'foo',
 notifySomebody: 1
 useLogging: 0
}
```

- a nested array:

```
{a : {
        a1 : "bla1",
        a2 : "bla2"
  },
 b : "hallo"
}
```

- a variable reference (=an object accessor):

```
{blogTitle : blog.title,
 blogObject: blog
}
```

Note: All these array examples will result into an associative array. If you have to supply a non-associative, i.e. numerically-indexed array, you'll write `{0: 'foo', 1: 'bar', 2: 'baz'}`.

Passing Data to the View

You can pass arbitrary objects to the view, using `$this->view->assign($identifier, $object)` from within the controller. See the above paragraphs about Object Accessors for details how to use the passed data.

Layouts

In almost all web applications, there are many similarities between each page. Usually, there are common templates or menu structures which will not change for many pages.

To make this possible in Fluid, we created a layout system, which we will introduce in this section.

Writing a Layout

Every layout is placed in the *Resources/Private/Layouts* directory, and has the file ending of the current format (by default *.html*). A layout is a normal Fluid template file, except there are some parts where the actual content of the target page should be inserted:

```
<html>
<head><title>My fancy web application</title></head>
<body>
<div id="menu">... menu goes here ...</div>
<div id="content">
  <f:render section="content" />
</div>
</body>
</html>
```

With this tag, a section from the target template is rendered.

Using a Layout

Using a layout involves two steps:

- Declare which layout to use: `<f:layout name="..." />` can be written anywhere on the page (though we suggest to write it on top, right after the namespace declaration) - the given name references the layout.
- Provide the content for all sections used by the layout using the `<f:section>...</f:section>` tag: `<f:section name="content">...</f:section>`

For the above layout, a minimal template would look like the following:

```
<f:layout name="example.html" />

<f:section name="content">
  This HTML here will be outputted to inside the layout
</f:section>
```

Writing Your Own ViewHelper

As we have seen before, all output logic resides in View Helpers. This includes the standard control flow operators such as if/else, HTML forms, and much more. This is the concept which makes Fluid extremely versatile and extensible.

If you want to create a view helper which you can call from your template (as a tag), you just write a plain PHP class which needs to inherit from `Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper` (or its subclasses). You need to implement only one method to write a view helper:

```
public function render()
```

Rendering the View Helper

We refresh what we have learned so far: When a user writes something like `<blog:displayNews />` inside a template (and has imported the `blog` namespace to `Neos\Blog\ViewHelpers`), Fluid will automatically instantiate the class `Neos\Blog\ViewHelpers\DisplayNewsViewHelper`, and invoke the render() method on it.

This `render()` method should return the rendered content as string.

You have the following possibilities to access the environment when rendering your view helper:

- `$this->arguments` is an associative array where you will find the values for all arguments you registered previously.
- `$this->renderChildren()` renders everything between the opening and closing tag of the view helper and returns the rendered result (as string).
- `$this->templateVariableContainer` is an instance of `Neos\FluidAdaptor\Core\ViewHelper\TemplateVariableContainer`, with which you have access to all variables currently available in the template, and can modify the variables currently available in the template.

Note: If you add variables to the `TemplateVariableContainer`, make sure to remove every variable which you added again. This is a security measure against side-effects.

It is also not possible to add a variable to the TemplateVariableContainer if a variable of the same name already exists - again to prevent side effects and scope problems.

Implementing a `for` ViewHelper

Now, we will look at an example: How to write a view helper giving us the `foreach` functionality of PHP.

A loop could be called within the template in the following way:

```
<f:for each="{blogPosts}" as="blogPost">
  <h2>{blogPost.title}</h2>
</f:for>
```

So, in words, what should the loop do?

It needs two arguments:

- `each`: Will be set to some object or array which can be iterated over.
- `as`: The name of a variable which will contain the current element being iterated over

It then should do the following (in pseudo code):

```
foreach ($each as $$as) {
  // render everything between opening and closing tag
}
```

Implementing this is fairly straightforward, as you will see right now:

```
class ForViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper {

  /**
   * Renders a loop
   *
   * @param array $each Array to iterate over
   * @param string $as Iteration variable
   */
  public function render(array $each, $as) {
        $out = '';
        foreach ($each as $singleElement) {
          $this->variableContainer->add($as, $singleElement);
          $out .= $this->renderChildren();
          $this->variableContainer->remove($as);
        }
        return $out;
  }

}
```

- The PHPDoc is part of the code! Fluid extracts the argument data types from the PHPDoc.
- You can simply register arguments to the view helper by adding them as method arguments of the `render()` method.
- Using `$this->renderChildren()`, everything between the opening and closing tag of the view helper is rendered and returned as string.

Declaring Arguments

We have now seen that we can add arguments just by adding them as method arguments to the `render()` method. There is, however, a second method to register arguments.

You can also register arguments inside a method called `initializeArguments()`. Call `$this->registerArgument($name, $dataType, $description, $isRequired, $defaultValue=NULL)` inside.

It depends how many arguments a view helper has. Sometimes, registering them as `render()` arguments is more beneficial, and sometimes it makes more sense to register them in `initializeArguments()`.

AbstractTagBasedViewHelper

Many view helpers output an HTML tag - for example `<f:link.action ...>` outputs a `<a href="...">` tag. There are many ViewHelpers which work that way.

Very often, you want to add a CSS class or a target attribute to an `<a href="...">` tag. This often leads to repetitive code like below. (Don't look at the code too thoroughly, it should just demonstrate the boring and repetitive task one would have without the `AbstractTagBasedViewHelper`):

```
class ActionViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper {

  public function initializeArguments() {
        $this->registerArgument('class', 'string', 'CSS class to add to the link');
        $this->registerArgument('target', 'string', 'Target for the link');
        ... and more ...
  }

  public function render() {
        $output = '<a href="..."';
        if ($this->arguments['class']) {
          $output .= ' class="' . $this->arguments['class'] . '"';
        }
        if ($this->arguments['target']) {
          $output .= ' target="' . $this->arguments['target'] . '"';
        }
        $output .= '>';
        ... and more ...
        return $output;
  }

}
```

Now, the `AbstractTagBasedViewHelper` introduces two more methods you can use inside `initializeArguments()`:

- `registerTagAttribute($name, $type, $description, $required)`: Use this method to register an attribute which should be directly added to the tag.
- `registerUniversalTagAttributes()`: If called, registers the standard HTML attributes `class`, `id`, `dir`, `lang`, `style`, `title`.

Inside the `AbstractTagBasedViewHelper`, there is a `TagBuilder` available (with `$this->tag`) which makes building a tag a lot more straightforward.

With the above methods, the `Link\ActionViewHelper` from above can be condensed as follows:

```
class ActionViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper {

        public function initializeArguments() {
                $this->registerUniversalTagAttributes();
        }

        /**
         * Render the link.
         *
         * @param string $action Target action
         * @param array $arguments Arguments
         * @param string $controller Target controller. If NULL current␣
↪controllerName is used
```

(continues on next page)

(continued from previous page)

```
         * @param string $package Target package. if NULL current package is used
         * @param string $subpackage Target subpackage. if NULL current subpackage is␣
↪used
         * @param string $section The anchor to be added to the URI
         * @return string The rendered link
         */
        public function render($action = NULL, array $arguments = array(),
                               $controller = NULL, $package = NULL, $subpackage =␣
↪NULL,
                                    $section = '') {
                $uriBuilder = $this->controllerContext->getURIBuilder();
                $uri = $uriBuilder->uriFor($action, $arguments, $controller, $package,
↪ $subpackage, $section);
                $this->tag->addAttribute('href', $uri);
                $this->tag->setContent($this->renderChildren());

                return $this->tag->render();
        }

}
```

Additionally, we now already have support for all universal HTML attributes.

Tip: The `TagBuilder` also makes sure that all attributes are escaped properly, so to decrease the risk of Cross-Site Scripting attacks, make sure to use it when building tags.

additionalAttributes

Sometimes, you need some HTML attributes which are not part of the standard. As an example: If you use the Dojo JavaScript framework, using these non-standard attributes makes life a lot easier.

We think that the templating framework should not constrain the user in his possibilities – thus, it should be possible to add custom HTML attributes as well, if they are needed. Our solution looks as follows:

Every view helper which inherits from `AbstractTagBasedViewHelper` has a special argument called `additionalAttributes` which allows you to add arbitrary HTML attributes to the tag.

If the link tag from above needed a new attribute called `fadeDuration`, which is not part of HTML, you could do that as follows:

```
<f:link.action action="..." additionalAttributes="{fadeDuration : 800}">
        Link with fadeDuration set
</f:link.action>
```

This attribute is available in all tags that inherit from `Neos\FluidAdaptor\Core\ViewHelper\AbstractTagBasedViewHelper`.

AbstractConditionViewHelper

If you want to build some kind of `if/else` condition, you should base the ViewHelper on the `AbstractConditionViewHelper`, as it gives you convenient methods to render the `then` or `else` parts of a ViewHelper. Let's look at the `<f:if>`-ViewHelper for a usage example, which should be quite self-explanatory:

```
class IfViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\
↪AbstractConditionViewHelper {

        /**
         * renders <f:then> child if $condition is true, otherwise renders <f:else>␣
↪child.
         *
         * @param boolean $condition View helper condition
         * @return string the rendered string
         */
        public function render($condition) {
                if ($condition) {
                        return $this->renderThenChild();
                } else {
                        return $this->renderElseChild();
                }
        }

}
```

By basing your condition ViewHelper on the `AbstractConditionViewHelper`, you will get the following features:

- Two API methods `renderThenChild()` and `renderElseChild()`, which should be used in the `then` / `else` case.
- The ViewHelper will have two arguments defined, called `then` and `else`, which are very helpful in the Inline Notation.
- The ViewHelper will automatically work with the `<f:then>` and `<f:else>`-Tags.

Widgets

Widgets are special ViewHelpers which encapsulate complex functionality. It can be best understood what widgets are by giving some examples:

- `<f:widget.paginate>` renders a paginator, i.e. can be used to display large amounts of objects. This is best known from search engine result pages.
- `<f:widget.autocomplete>` adds autocompletion functionality to a text field.
- More widgets could include a Google Maps widget, a sortable grid, ...

Internally, widgets consist of an own Controller and View.

Using Widgets

Using widgets inside your templates is really simple: Just use them like standard ViewHelpers, and consult their documentation for usage examples. An example for the `<f:widget.paginate>` follows below:

```
<f:widget.paginate objects="{blogs}" as="paginatedBlogs" configuration="
↪{itemsPerPage: 10}">
  // use {paginatedBlogs} as you used {blogs} before, most certainly inside
  // a <f:for> loop.
</f:widget.paginate>
```

In the above example, it looks like `{blogs}` contains all `Blog` objects, thus you might wonder if all objects were fetched from the database. However, the blogs are *not fetched* from the database until you actually use them, so the Paginate Widget will adjust the query sent to the database and receive only the small subset of objects.

So, there is no negative performance overhead in using the Paginate Widget.

Writing widgets

We already mentioned that a widget consists of a controller and a view, all triggered by a ViewHelper. We'll now explain these different components one after each other, explaining the API you have available for creating your own widgets.

ViewHelper

All widgets inherit from `Neos\FluidAdaptor\Core\Widget\AbstractWidgetViewHelper`. The ViewHelper of the widget is the main entry point; it controls the widget and sets necessary configuration for the widget.

To implement your own widget, the following things need to be done:

- The controller of the widget needs to be injected into the `$controller` property.
- Inside the `render()`-method, you should call `$this->initiateSubRequest()`, which will initiate a request to the controller which is set in the `$controller` property, and return the `Response` object.
- By default, all ViewHelper arguments are stored as *Widget Configuration*, and are also available inside the Widget Controller. However, to modify the Widget Configuration, you can override the `getWidgetConfiguration()` method and return the configuration which you need there.

There is also a property `$ajaxWidget`, which we will explain later in *Ajax Widgets*.

Controller

A widget contains one controller, which must inherit from `Neos\FluidAdaptor\Core\Widget\AbstractWidgetController`, which is an `ActionController`. There is only one difference between the normal `ActionController` and the `AbstractWidgetController`: There is a property `$widgetConfiguration`, containing the widget's configuration which was set in the ViewHelper.

Fluid Template

The Fluid templates of a widget are normal Fluid templates as you know them, but have a few ViewHelpers available additionally:

<f:widget.uri> Generates an URI to another action of the widget.

<f:widget.link> Generates a link to another action of the widget.

<f:renderChildren> Can be used to render the child nodes of the Widget ViewHelper, possibly with some more variables declared.

Ajax Widgets

Widgets have special support for AJAX functionality. We'll first explain what needs to be done to create an AJAX compatible widget, and then explain it with an example.

To make a widget AJAX-aware, you need to do the following:

- Set `$ajaxWidget` to TRUE inside the ViewHelper. This will generate an unique AJAX Identifier for the Widget, and store the WidgetConfiguration in the user's session on the server.
- Inside the index-action of the Widget Controller, generate the JavaScript which triggers the AJAX functionality. There, you will need a URI which returns the AJAX response. For that, use the following ViewHelper inside the template:

```
<f:widget.uri ajax="TRUE" action="..." arguments="..." />
```

- Inside the template of the AJAX request, `<f:renderChildren>` is not available, because the child nodes of the Widget ViewHelper are not accessible there.

XSD schema generation

A XSD schema file for your ViewHelpers can be created by executing

```
./flow documentation:generatexsd <Your>\\<Package>\\ViewHelpers
        --target-file /some/directory/your.package.xsd
```

Then import the XSD file in your favorite IDE and map it to the namespace `http://neos.io/ns/<Your/Package>/ViewHelpers`. Add the namespace to your Fluid template by adding the `xmlns` attribute to the root tag (usually `<xml ...>` or `<html ...>`).

Note: You are able to use a different XML namespace pattern by specifying the `--xsd-namespace argument` in the generatexsd command.

If you want to use this inside partials, you can use the "section" argument of the render ViewHelper in order to only render the content of the partial.

Partial:

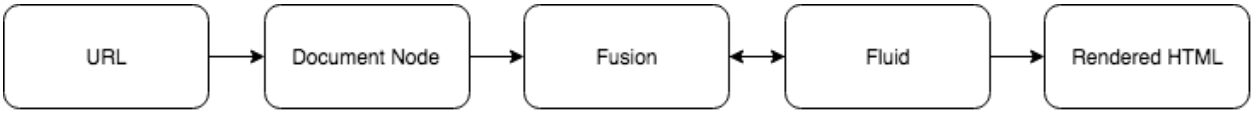
```
<html xmlns:x="http://neos.io/ns/Your/Package/ViewHelpers">
<f:section name="content">
        <x:yourViewHelper />
</f:section>
```

Template:

```
<f:render partial="PartialName" section="content" />
```

3.3.2 Rendering A Page

This section explains how pages are rendered in Neos. More precisely, we show how to render a node of type `Neos.Neos:Document`. The default page type in Neos (`Neos.NodeTypes:Page`) inherits from this type. If you create custom document node types, they need to be a subtype of `Neos.Neos:Document` as well. This section also explains how to implement custom rendering for your own document node types.



1. An URL is requested from Neos through an HTTP request.
2. The requested URL is resolved to a node. This works via the Frontend `NodeController` and the `NodeConverter` of the Neos CR by translating the URL path to a node path, and then finding the node with this path. The document node resolution is completely done in the Neos core - usually, site integrators do not need to modify it.
3. The document node is passed to Fusion, which is the Neos rendering engine. Rendering always starts at the Fusion path `root`. This rendering process is explained in detail below.
4. Fusion can render Fluid templates, which in turn can call Fusion again to render parts of themselves. This can go back and forth multiple times, even recursively.
5. Once Fusion has traversed the rendering tree fully, rendering is done and the rendered output (usually HTML, but Fusion can render arbitrary text formats) is sent back to the requester.

The `root` path

You may already have seen a `Root.fusion` that contain a path `page` which is filled with an object of type `Neos.Neos:Page`. Here, the `Neos.Neos:Page` Fusion object is assigned to the path `page`, telling the system that the Fusion object `Page` is responsible for further rendering:

```
page = Neos.Neos:Page {
  head {
    [...]
  }
  body {
    [...]
  }
}
```

Let's investigate how this rendering process happens. Fusion always starts rendering at the fusion path `root`. You can verify this by simply replacing the code in your `Root.fusion` file with this snippet:

```
root = "Hello World!"
```

All page rendering will disappear and only the words "Hello World" will be rendered by Neos.

Using the `page` path is not the recommended way to render your document node types anymore. We encourage you to define a prototype named after your document node type extending `Neos.Neos:Page`. Read *Rendering Custom Document Types* for further details and how to achieve this.

The root `Neos.Fusion:Case` object

The `root` path contains, by default, a `Neos.Fusion:Case` object. Here is a section from this object - to see the full implementation, check out the file `DefaultFusion.fusion` in the package `Neos.Neos`, path `Resources\Private\Fusion`.

```
root = Neos.Fusion:Case {

  [...more matchers before...]

  documentType {
    condition = Neos.Fusion:CanRender {
      type = ${q(documentNode).property('_nodeType.name')}
    }
    type = ${q(documentNode).property('_nodeType.name')}
  }

  default {
    condition = TRUE
    renderPath = '/page'
  }
}
```

If you do not know what a `Case` object does, you might want to have a look at the *Fusion Reference*. All paths in the `Case` object (so-called *matchers*) check a certain condition - the `condition` path in the matcher. Matchers are evaluated one after another, until one condition evaluates to `TRUE`. If it does, matcher's `type`, `renderer` or `renderPath` path (whichever exists) will be evaluated. If no other condition matches, the `default` matcher is evaluated and points Fusion to the path `page`. Rendering then continues with the `page` path, which is by default generated in your site package's `Root.fusion` file. This is why, if you don't do anything else, rendering begins at your `page` path.

The current best practice is to use the `documentType` matcher by defining your own Fusion prototypes for each document type. This approach will be covered further below.

The page path and `Neos.Neos:Page` object

The minimally needed Fusion for rendering a page looks as follows:

```
page = Page {
  body {
    templatePath = 'resource://My.Package/Private/Templates/PageTemplate.html'
  }
}
```

`Page` expects one parameter to be set: The path of the Fluid template which is rendered inside the `<body>` of the resulting HTML page.

If the template above is an empty file, the output shows how minimal Neos impacts the generated markup:

```
<!DOCTYPE html>
<html>
  <!--
      This website is powered by Neos, the Open Source Content Application Platform␣
↪licensed under the GNU/GPL.
      Neos is based on Flow, a powerful PHP application framework licensed under the␣
↪MIT license.
```

(continues on next page)

(continued from previous page)

```
      More information and contribution opportunities at https://www.neos.io
  -->
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <script src="/_Resources/Static/Packages/Neos.Neos/JavaScript/LastVisitedNode.js"␣
↪data-neos-node="a319a653-ef38-448d-9d19-0894299068aa"></script>
  </body>
</html>
```

It becomes clear that Neos gives as much control over the markup as possible to the integrator: No body markup, no styles, only little Javascript to record the last visited page to redirect back to it after logging in. Except for the charset meta tag nothing related to the content is output by default.

If the template file is filled with the following content:

```
<h1>{title}</h1>
```

the body would contain a heading to output the title of the current page:

```
<body>
  <h1>My first page</h1>
</body>
```

Again, no added CSS classes, no wraps. Why `{title}` outputs the page title is covered in detail below.

Adding pre-rendered output to the page template

Of course the current template is still quite boring; it does not show any content or any menu. In order to change that, the Fluid template is adjusted as follows:

```
{namespace fusion=Neos\Fusion\ViewHelpers}
{parts.menu -> f:format.raw()}
<h1>{title}</h1>
{content.main -> f:format.raw()}
```

Placeholders for the menu and the content have been added. Because the `parts.menu` and `content.main` refer to a rendered Fusion path, the output needs to be passed through the `f:format.raw()` ViewHelper. The Fusion needs to be adjusted as well:

```
page = Neos.Neos:Page {
  body {
    templatePath = 'resource://My.Package/Private/Templates/PageTemplate.html'

    parts {
      menu = Neos.Neos:Menu
    }

    content {
      main = Neos.Neos:PrimaryContent {
        nodePath = 'main'
      }
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

In the above Fusion, a Fusion object at `page.body.parts.menu` is defined to be of type `Neos.Neos:Menu`. It is exactly this Fusion object which is rendered, by specifying its relative path inside `{parts.menu -> f:format.raw()}`.

Furthermore, the `Neos.Neos:PrimaryContent` Fusion object is used to render a Neos ContentRepository `ContentCollection` node. Through the `nodePath` property, the name of the Neos ContentRepository `ContentCollection` node to render is specified. As a result, the web page now contains a menu and the contents of the main content collection.

The use of `content` and `parts` here is **just a convention**, the names can be chosen freely. In the example `content` is used for the section where content is later placed, and `parts` is for anything that is not *content* in the sense that it will directly be edited in the content module of Neos.

The `Neos.Neos:Page` object in more detail

To understand what the `Neos.Neos:Page` object actually does, it makes sense to look at its definition. We can find the `Page` prototype in the file `Page.fusion` in the path `Resources\Private\Fusion` inside the `Neos.Neos` package. Here is a snippet taken from this object's definition:

```
prototype(Neos.Neos:Page) < prototype(Neos.Fusion:Http.Message) {

  # The content of the head tag, integrators can add their own head content in this␣
↪array.
  head = Neos.Fusion:Array {
    # Link tags for stylesheets in the head should go here
    stylesheets = Neos.Fusion:Array

    # Script includes in the head should go here
    javascripts = Neos.Fusion:Array {
      @position = 'after stylesheets'
    }
  }

  # Content of the body tag. To be defined by the integrator.
  body = Neos.Fusion:Template {
    node = ${node}
    site = ${site}

    # Script includes before the closing body tag should go here
    javascripts = Neos.Fusion:Array

    # This processor appends the rendered javascripts Array to the rendered template
    @process.appendJavaScripts = ${value + this.javascripts}
  }
}
```

By looking at this definition, we understand a bit more about how page rendering actually works. `Neos.Neos:Page` inherits from `Neos.Fusion:Http.Message`, which in turn inherits from `Neos.Fusion:Array`. `Array` fusion objects just render their keys one after another, so the Page object just outputs whatever is in it. The `Neos.Neos:Page` object renders the HTML framework, such as doctype, head and body tags, and also defines the default integration points for site integrators - `head` and `body` as well as their inner objects. It is not by coincidence that these exact paths are pre-filled with sensible defaults in site package's generated default `Root.fusion` files.

We can also see that the `body` object is a `Neos.Fusion:Template`, which is why we have to set the template path to a Fluid template which will be rendered as the body.

Rendering custom document types

There are two basic approaches to render different document types. We currently recommend to create a Fusion prototype per custom page type, which is since Neos 4.0 automatically picked up by Neos (see below). The "old" way involves adding one root matcher per document type, explicitly checking for the node type in the condition, and redirecting Fusion to another render path. It is documented here for completeness' sake, but we do not recommend to use it anymore.

Prototype-based rendering

Since Neos 4.0, the root `Case` object ships with a `documentType` matcher, which will automatically pick up and render Fusion prototypes with the same name as the corresponding document node type, if they exist. This snippet of Fusion in the root `Case` is responsible for it:

```
root = Neos.Fusion:Case {

  [...]

  documentType {
    condition = Neos.Fusion:CanRender {
      type = ${q(documentNode).property('_nodeType.name')}
    }
    type = ${q(documentNode).property('_nodeType.name')}
  }

  [...]
}
```

This means that if you have a custom page type `Your.Site:CustomPage`, you simply have to create a Fusion prototype with a matching name to get different rendering for it. We explain how to do this in more detail in the "How To" section of the docs: *Rendering Custom Document Types*

Explicit path rendering (discouraged)

Before document-based rendering, you had to add your own matchers to the root object to get different rendering:

```
root.customPageType1 {
  condition = ${q(node).is('[instanceof Your.Site:CustomPage]')}
  renderPath = '/custom1'
}

custom1 < page
custom1 {
  # output modified here...
}
```

There are a number of disadvantages of doing this, which is why we recommend to stick to prototype-based rendering:

- We are polluting the `root` namespace, adding to the danger of path collision
- We need to copy and modify the `page` object for each new document type, which becomes messy

- The order of path copying is important, therefore introducing possibly unwanted side effects

Further Reading

Details on how Fusion works and can be used can be found in the section *Inside Fusion*. *Adjusting Neos Output* shows how page, menu and content markup can be adjusted freely.

3.3.3 Creating Custom Content Elements

Neos ships with commonly used, predefined content elements, but it is easily possible to amend and even completely replace them.

Defining new content elements is usually a three-step process:

1. Defining a *Neos ContentRepository Node Type*, listing the properties and types of the node.
2. Defining a *Fusion object* which is responsible for rendering this content type. Usually, this is a wrapper for a Fluid Template which then defines the rendered markup.
3. Add a *Fluid Template* which contains the markup being rendered

Creating a Simple Content Element

The following example creates a new content element *Acme.Demo:YouTube* which needs the YouTube URL and then renders the video player.

First, the *Neos ContentRepository Node Type* needs to be defined in *NodeTypes.yaml*. This can be done in your site package or in a package dedicated to content elements, if reuse is foreseeable.

```
'Acme.Demo:YouTube':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    group: 'general'
    label: 'YouTube Video'
    inspector:
      groups:
        video:
          label: 'Video'
          icon: 'icon-film'
  properties:
    videoUrl:
      type: string
      ui:
        label: 'Video URL'
        reloadIfChanged: TRUE
        inspector:
          group: 'video'
```

The declaration of node types with all required and optional properties is documented in *Node Type Definition*.

Next the Fusion rendering for the content element has to be defined. By convention, a Fusion object with the same name as the content element is used for rendering; thus in this case a Fusion object *My.Package:YouTube*:

```
prototype(Acme.Demo:YouTube) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://Acme.Demo/Private/Templates/FusionObjects/YouTube.
↪html'
        videoUrl = ${q(node).property('videoUrl')}
        width = '640'
        height = '360'
}
```

A new Fusion object prototype with the name *My.Package:YouTube* is declared, inheriting from the pre-defined *Template* Fusion object which provides rendering through Fluid.

The *templatePath* property of the *YouTube* Fusion object is set to point to the Fluid template to use for rendering. All (other) properties which are set on the *Template* Fusion object are directly made available inside Fluid as variables – and because the *YouTube* Fusion object extends the *Template* Fusion object, this rule also applies there.

Thus, the last line defines a *videoUrl* variable to be available inside Fluid, which is set to the result of the Eel expression *${q(node).property('videoUrl')}*. Eel is explained in depth in *Eel, FlowQuery and Fizzle*, but this is a close look at the used expression *q(node).property('videoUrl')*:

- The q() function wraps its argument, in this case the Neos ContentRepository Node which is currently rendered, into *FlowQuery*.
- FlowQuery defines the *property(...)* operation used to access the property of a node.

To sum it up: The expression *${q(node).property('videoUrl')}* is an Eel expression, in which FlowQuery is called to return the property *videoUrl* of the current node.

The final step in creating the YouTube content element is defining the *YouTube.html* Fluid template, f.e. with the following content:

```
<iframe width="{width}" height="{height}" src="{videoUrl}" frameborder="0"␣
↪allowfullscreen></iframe>
```

In the template the *{videoUrl}* variable which has been defined in Fusion is used as we need it.

What are the benefits of indirection through Fusion?

In the above example the *videoUrl* property of the *Node* is not directly rendered inside the Fluid template. Instead *Fusion* is used to pass the *videoUrl* from the *Node* into the Fluid template.

While this indirection might look superfluous at first sight, it has important benefits:

- The Fluid Template does not need to know anything about *Nodes*. It just needs to know that it outputs a certain property, but not where it came from.
- Because the rendering is decoupled from the data storage this way, the Fusion object can be instantiated directly, manually setting a *videoUrl*:

  ```
  page.body.parts.teaserVideo = My.Package:YouTube {
    videoUrl = 'http://youtube.com/.....'
  }
  ```

- If a property needs to be modified *just slightly*, a *processor* can be used for declarative modification of this property in Fusion; not even touching the Fluid template. This is helpful for smaller adjustments to foreign packages.

Creating Editable Content Elements

The simple content element created in *Creating a Simple Content Element* exposes the video URL only through the property inspector in the editing interface. Since the URL is not directly visible this is the only viable way.

In case of content that is directly visible in the output, inline editing can be enabled by slight adjustments to the process already explained.

The node type definition must define which properties are inline editable through setting the *inlineEditable* property:

```
'Acme.Demo:Quote':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    group: 'general'
    label: 'Quote'
  properties:
    quote:
      type: string
      defaultValue: 'Use the force, Luke!'
      ui:
        label: 'Quote'
        inlineEditable: TRUE
```

The Fusion for the content element is the same as for a non-inline-editable content element:

```
prototype(Acme.Demo:Quote) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://Acme.Demo/Private/Templates/FusionObjects/Quote.
↪html'
        quote = ${q(node).property('quote')}
}
```

The Fluid template again needs some small adjustment in form of the *contentElement.editable* ViewHelper to declare the property that is editable. This may seem like duplication, since the node type already declares the editable properties. But since in a template multiple editable properties might be used, this still is needed.

```
{namespace neos=Neos\Neos\ViewHelpers}
<blockquote>
        {neos:contentElement.editable(property: 'quote')}
</blockquote>
```

The `blockquote` is wrapped around the *contentElement.editable* and not the other way because that would mean the blockquote becomes a part of the editable content, which is not desired in this case.

Using the *tag* attribute to make the ViewHelper use the `blockquote` tag needed for the element avoids the nesting in an additional container *div* and thus cleans up the generated markup:

```
{namespace neos=Neos\Neos\ViewHelpers}
{neos:contentElement.editable(property: 'quote', tag: 'blockquote')}
```

A property can be inline editable *and* appear in the property inspector if configured accordingly. In such a case *reloadIfChanged* should be enabled to make changes in the property editor visible in the content area.

Creating Nested Content Elements

In case content elements do not only contain simple properties, but arbitrary sub-elements, the process again is roughly the same. To demonstrate this, a *Video Grid* content element will be created, which can contain two texts and two videos.

1. A Neos ContentRepository Node Type definition is created. It makes use of the *childNodes* property to define (and automatically create) sub-nodes when a node of this type is created. In the example the two video and text elements will be created directly upon element creation:

```
'Acme.Demo:VideoGrid':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    group: 'structure'
    label: 'Video Grid'
  childNodes:
    video0:
      type: 'Acme.Demo:YouTube'
    video1:
      type: 'Acme.Demo:YouTube'
    text0:
      type: 'Neos.NodeTypes:Text'
    text1:
      type: 'Neos.NodeTypes:Text'
```

2. The needed Fusion is created:

```
prototype(Acme.Demo:VideoGrid) {
        videoRenderer = Acme.Demo:YouTube
        textRenderer = Neos.NodeTypes:Text

        video0 = ${q(node).children('video0').get(0)}
        video1 = ${q(node).children('video1').get(0)}

        text0 = ${q(node).children('text0').get(0)}
        text1 = ${q(node).children('text1').get(0)}
}
```

   Instead of assigning variables to the Fluid template, *additional Fusion objects* responsible for the video and the text rendering are instantiated. Furthermore, the video and text nodes are fetched using Eel and then passed to the Fluid template.

3. The Fluid template is created. Instead of outputting the content directly using object access on the passed nodes, the *<ts:render>* ViewHelper is used to defer rendering to Fusion again. The needed Neos ContentRepository Node is passed as context to Fusion:

```
{namespace fusion=Neos\Fusion\ViewHelpers}
<fusion:render path="videoRenderer" context="{node: video0}" />
<fusion:render path="textRenderer" context="{node: text0}" />
<br />
<fusion:render path="videoRenderer" context="{node: video1}" />
<fusion:render path="textRenderer" context="{node: text1}" />
```

Instead of referencing specific content types directly the use of the generic *ContentCollection* content element allows to insert *arbitrary content* inside other elements. An example can be found in the *Neos.NodeTypes:MultiColumn* and *Neos.NodeTypes:MultiColumnItem* content elements.

As explained earlier (in *What are the benefits of indirection through Fusion?*) the major benefit if using Fusion to decouple the rendering of items this way is flexibility. In the video grid it shows how this enables *composability*, other Fusion objects can be re-used for rendering smaller parts of the element.

Content Element Group

In Neos content elements are grouped by type. By default the following groups are available:

general Basic content elements, like *text* and *image*.

structure Elements defining a structure. This group contains for example the 2 column element.

plugins Available plugins in the site installation.

It is possible to create new groups by using the *Neos.Neos.nodeTypes.groups* settings. Registering 2 new groups could look like:

```
Neos:
  Neos:
    nodeTypes:
      groups:
        form:
          label: 'Form elements'
        special:
          position: 50
          label: 'Special elements'
          collapsed: true
          icon: 'icon-fort-awesome'
```

The groups are ordered by the position argument.

Extending The Inspector

Warning: Adding editors and validators is no fixed API yet, keep an eye on the changelogs if you use this.

It is possible to extend the inspector for adding new editors and validators to edit the properties of your nodetypes.

Editors

By default the following list of editors is available in Neos:

- *Neos.Neos/Inspector/Editors/BooleanEditor*

  A checkbox, by default configured for properties of type *boolean*.

- *Neos.Neos/Inspector/Editors/DateTimeEditor*

  A datepicker with support for time selection too. By default configured for properties of type *date*.

- *Neos.Neos/Inspector/Editors/CodeEditor*

  An code editor with syntax highlighting. You can use this editor for editing other types of *textual* content, by configuring a different *highlightingMode* and *buttonLabel* to change usage for this editor:

```
style:
  type: string
  ui:
    label: 'CSS'
    reloadIfChanged: TRUE
    inspector:
      group: 'code'
      editor: 'Neos.Neos/Inspector/Editors/CodeEditor'
      editorOptions:
        buttonLabel: 'Edit CSS source'
        highlightingMode: 'text/css'
```

- *Neos.Neos/Inspector/Editors/ImageEditor*

  An image editor with cropping and size support. By default configured for properties of type *NeosMediaDomainModelImageInterface*.

- *Neos.Neos/Inspector/Editors/ReferenceEditor*

  A selector with autocomplete to reference to another node. By default configured for properties of type *reference*.

- *Neos.Neos/Inspector/Editors/ReferencesEditor*

  A selector with autocomplete to reference to multiple nodes. By default configured for properties of type *references*.

- *Neos.Neos/Inspector/Editors/SelectBoxEditor*

  A selectbox.

- *Neos.Neos/Inspector/Editors/TextFieldEditor*

  A simple textfield. By default configured for properties of type *string* and *integer*

The following editors are also available, but will most likely only be used internally in Neos:

- *Neos.Neos/Inspector/Editors/MasterPluginEditor*
- *Neos.Neos/Inspector/Editors/PluginViewEditor*
- *Neos.Neos/Inspector/Editors/PluginViewsEditor*

Register Custom Editors

There are 2 ways to register custom editors. Either by registering a namespace for a group of editors, or by selecting the direct path to an editor specifically.

Registering a namespace pointing to a folder containing editors works as follows:

- Create a folder containing the JavaScript sources for the editors
- Name your files *PropertyTypeEditor*
- Configure the path as a requirejs path mapping using the following Settings.yaml

```
Neos:
  Neos:
    userInterface:
      requireJsPathMapping:
        'My.Package/Inspector/Editors': 'resource://My.Package/Public/Scripts/
↪Path/To/Folder'
```

- Now configure the editor for your property in the NodeTypes.yaml:

```
'My.Package:NodeType':
  properties:
    myProperty:
      type: 'string'
      ui:
        inspector:
          editor: 'My.Package/Inspector/Editors/PropertyTypeEditor'
          editorOptions:
            optionName: 'optionValue'
```

To set global options for your editor you can set a set of defaults in Settings.yaml:

```
Neos:
  Neos:
    userInterface:
      inspector:
        editors:
          'My.Package/Inspector/Editors/PropertyTypeEditor':
            editorOptions:
              optionName: 'optionValue'
```

The editor options set on a property level will override the global editor options.

To register just one specific path as an editor use the following code:

```
Neos:
  Neos:
    userInterface:
      inspector:
        editors:
          'My.Package/Inspector/Editors/CustomEditor':
            path: 'resource://My.Package/Public/Scripts/Path/To/File/Without/Js/
↪Extension'
```

Validators

By default the following validators are available in Neos:

- *Neos.Neos/Validation/AbstractValidator*

  This *abstract* validator should be used to base custom validators on.

- *Neos.Neos/Validation/AlphanumericValidator*

  Supported options:

  - regularExpression

- *Neos.Neos/Validation/CountValidator*

  Supported options:

  - minimum
  - maximum

- *Neos.Neos/Validation/DateTimeRangeValidator*

  Supported options:

  - latestDate
  - earliestDate
- *Neos.Neos/Validation/DateTimeValidator*
- *Neos.Neos/Validation/EmailAddressValidator*

  Supported options:
  - regularExpression
- *Neos.Neos/Validation/FloatValidator*
- *Neos.Neos/Validation/IntegerValidator*
- *Neos.Neos/Validation/LabelValidator*

  Supported options:
  - regularExpression
- *Neos.Neos/Validation/NumberRangeValidator*

  Supported options:
  - minimum
  - maximum
- *Neos.Neos/Validation/RegularExpressionValidator*

  Supported options:
  - regularExpression
- *Neos.Neos/Validation/StringLengthValidator*

  Supported options:
  - minimum
  - maximum
- *Neos.Neos/Validation/StringValidator*
- *Neos.Neos/Validation/TextValidator*
- *Neos.Neos/Validation/UuidValidator*

  Supported options:
  - regularExpression

Register Custom Validators

There are 2 ways to register custom validators. Either by registering a namespace for a group of validators, or by selecting the direct path to an validator specifically.

Registering a namespace pointing to a folder containing validators works as follows:

- Create a folder containing the JavaScript sources for the validators
- Name your files *DataTypeValidator*
- Configure the path as a requirejs path mapping using the following Settings.yaml

```
Neos:
  Neos:
    userInterface:
      requireJsPathMapping:
        'My.Package/Validation': 'resource://My.Package/Public/Scripts/Path/To/
↪Folder'
```

- Now configure the validator for your property in the NodeTypes.yaml:

```
'My.Package:NodeType':
  properties:
    myProperty:
      type: 'string'
      validation:
        'My.Package/Validation/DataTypeValidator': []
```

To register just one specific path as a validator use the following code:

```
Neos:
  Neos:
    userInterface:
      validators:
        'My.Package/Validation/CustomValidator':
          path: 'resource://My.Package/Public/Scripts/Path/To/File/Without/Js/
↪Extension'
```

3.3.4 Adjusting Neos Output

Page Template

The page template defines the overall structure of the generated markup: what is rendered in the body and head of the resulting document.

The Body

As briefly explained in *Rendering A Page* the path to your own template for the `body` of a generated page can be set using Fusion:

```
page = Page
page.body.templatePath = 'resource://My.Package/Private/Templates/PageTemplate.html'
```

The file will the be used to render the body content and any Fluid placeholders will be substituted, ViewHelpers will be executed. Since no further information is given to the rendering process, the full content of the template will be used for the body.

If the template contains a full HTML page, this will lead to invalid markup. But in most cases having the template as a full HTML document is desired, as it allows easy handling by the developer and can be previewed as is in a browser.

To use just a part of the document for the body, that part can simply be enclosed in a Fluid section:

```
<!DOCTYPE html>
<html>
<head>
        ...
```

(continues on next page)

(continued from previous page)

```
</head>
<body>
<f:section name="body">
        <h1>{title}</h1>
</f:section>
</body>
</html>
```

The Fusion is then amended with the declaration of the section to use:

```
page = Page
page.body {
        templatePath = 'resource://My.Package/Private/Templates/PageTemplate.html'
        sectionName = 'body'
}
```

This results in only the part inside the template's "body" section to be used for rendering the body of the generated page.

To add actual content from Neos to the desired places in the markup, a special ViewHelper to turn control back to Fusion is used. This has been mentioned in *Rendering A Page* already.

This template uses the `render` ViewHelper twice, once to render the path *parts/menu* and once to render the path *content.main*:

```
<f:section name="body">
        <ts:render path="parts.menu" />
        <h1>{title}</h1>
        <ts:render path="content.main" />
</f:section>
```

Those paths are relative to the current path. Since that part of the template is rendered by the Fusion object at *page.body*, this is the starting point for the relative paths. This means the `Menu` and the `ContentCollection` in this Fusion are used for rendering the output:

```
page = Page
page.body {
        templatePath = 'resource://My.Package/Private/Templates/PageTemplate.html'
        sectionName = 'body'
        parts.menu = Menu
        content.main = ContentCollection
        content.main.nodePath = 'main'
}
```

The Head

The `head` of a page generated by Neos contains only minimal content by default. Apart from the meta tag declaring the character set it is empty:

```
<head>
        <meta charset="UTF-8" />
</head>
```

To fill this with life, it is recommended to add sections to the head of your HTML template that group the needed parts. Additional Fusion *Template* objects are then used to include them into the generated page. Here is an example:

Page/Default.html

```
<head>
        <f:section name="meta">
                <title>{title}</title>
        </f:section>

        <f:section name="stylesheets">
                <!-- put your stylesheet inclusions here, they will be included in␣
↪your website by Fusion -->
        </f:section>

        <f:section name="scripts">
                <!-- put your javascript inclusions here, they will be included in␣
↪your website by Fusion -->
        </f:section>
</head>
```

Library/Root.fusion

```
page.head {
        meta = Neos.Fusion:Template {
                templatePath = 'resource://Acme.DemoCom/Private/Templates/Page/
↪Default.html'
                sectionName = 'meta'

                title = ${q(node).property('title')}
        }
        stylesheets.site = Neos.Fusion:Template {
                templatePath = 'resource://Acme.DemoCom/Private/Templates/Page/
↪Default.html'
                sectionName = 'stylesheets'
        }
        javascripts.site = Neos.Fusion:Template {
                templatePath = 'resource://Acme.DemoCom/Private/Templates/Page/
↪Default.html'
                sectionName = 'scripts'
        }
}
```

The Fusion fills the *page.head* instance of `Neos.Fusion:Array` with content. The predefined paths for *page.head.stylesheets*, *page.head.javascripts* or *page.body.javascripts* should be used to add custom includes. They are implemented by a Fusion *Array* and allow arbitrary items to specify JavaScript or CSS includes without any restriction on the content.

This will render some more head content:

```
        <head>
        ...
        <title>Home</title>
        <!-- put your stylesheet inclusions here, they will be included in your␣
↪website by Fusion -->
        <!-- put your javascript inclusions here, they will be included in your␣
↪website by Fusion -->
        ...
</head>
```

This provides for flexibility and allows to control precisely what ends up in the generated markup. Anything that is needed can be added freely, it just has to be in a section that is included.

Menu Rendering

Out of the box the *Menu* is rendered using a simple unsorted list:

```
<ul class="nav">
        <li class="current">
                <a href="home.html">Home</a>
        </li>

        <li class="normal">
                <a href="blog.html">Blog</a>
        </li>
</ul>
```

Wrapping this into some container (if needed) in a lot of cases provides for enough possibilities to style the menu using CSS. In case it still is needed, it is possible to change the rendered markup of *Menu* using Fusion. *Menu* is defined inside the core of Neos together with Neos.NodeTypes:

Neos.Neos/Resources/Private/Fusion/Root.fusion

```
prototype(Neos.Neos:Menu).@class = 'Neos\\Neos\\Fusion\\MenuImplementation'
```

Neos.NodeTypes/Resources/Private/Fusion/Root.fusion

```
prototype(Neos.NodeTypes:Menu) < prototype(Neos.Neos:Menu)
prototype(Neos.NodeTypes:Menu) {
        templatePath = 'resource://Neos.NodeTypes/Private/Templates/FusionObjects/
↪Menu.html'
        entryLevel = ${String.toInteger(q(node).property('startLevel'))}
        maximumLevels = ${String.toInteger(q(node).property('maximumLevels'))}
        node = ${node}
}
```

The above code defines the *prototype* of *Menu* with the *prototype(Menu)* syntax. This prototype is the "blueprint" of all *Menu* objects which are instantiated. All properties which are defined on the prototype (such as *@class* or *templatePath*) are automatically active on all *Menu instances*, if they are not explicitly overridden.

One way to adjust the menu rendering is to override the *templatePath* property, which points to a Fluid template. To achieve that, we have two possibilities.

First, the *templatePath* for the menu at *page.body.parts.menu* can be set:

```
page.body.parts.menu.templatePath = 'resource://My.Package/Private/Templates/
↪MyMenuTemplate.html'
```

This overrides the *templatePath* which was defined in *prototype(Menu)* for this single menu.

Second, the *templatePath* inside the *Menu* prototype itself can be changed:

```
prototype(Menu).templatePath = 'resource://My.Package/Private/Templates/
↪MyMenuTemplate.html'
```

In this case, the changed template path is used for *all menus* which do not override the *templatePath* explicitly. Every time *prototype(...)* is used, this can be understood as: "For *all* objects of type ..., define *something*"

After setting the path, changing the menu is simply a job of copying the default *Menu* template into *MyMenuTemplate.html* and adjusting the markup as needed.

Menu states

The default *Menu* implementation assigns CSS classes to the *li* tags depending on their state:

current A menu item pointing to the page that is currently shown

active Any menu item that is on the path to the *current* page

normal Any menu item that is neither *current* nor *active*

Content Element Rendering

The rendering of content elements follows the same principle as shown for the *Menu*. The default Fusion is defined in the Neos.NodeTypes package and the content elements all have default Fluid templates.

Combined with the possibility to define custom templates per instance or on the prototype level, this already provides a lot of flexibility. Another possibility is to inherit from the existing Fusion and adjust as needed using Fusion.

The available properties and settings that the Fusion objects in Neos provide are described in *Fusion Reference*.

Including CSS and JavaScript in a Neos Site

Including CSS and JavaScript should happen through one of the predefined places of the *Page* object. Depending on the desired position one of the *page.head.javascripts*, *page.head.stylesheets* or *page.body.javascripts* Arrays should be extended with an item that renders script or stylesheet includes:

```
page.head {

        stylesheets {
                bootstrap = '<link href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/
↪css/bootstrap.min.css" rel="stylesheet">'
        }

        javascripts {
                jquery = '<script src="//code.jquery.com/jquery-1.10.1.min.js"></
↪script>'
        }

}

page.body {

        javascripts {
                bootstrap = '<script src="//netdna.bootstrapcdn.com/bootstrap/3.0.3/
↪js/bootstrap.min.js"></script>'
        }

}
```

The *page.body.javascripts* content will be appended to the rendered page template so the included scripts should be placed before the closing body tag. As always in Fusion the elements can be a simple string value, a Fusion object like *Template* or an expression:

```
page.head {
        # Add a simple value as an item to the javascripts Array
        javascripts.jquery = '<script src="//code.jquery.com/jquery-1.10.1.min.js"></
↪script>'
```

(continues on next page)

(continued from previous page)

```
        # Use an expression to render a CSS include (this is just an example,␣
↪bootstrapVersion is not defined by Neos)
        stylesheets.bootstrap = ${'<link href="//netdna.bootstrapcdn.com/bootstrap/'␣
↪+ bootstrapVersion + '/css/bootstrap.min.css" rel="stylesheet">'}
}

page.body {
        # Use a Template object to access a special section of the site template
        javascripts.site = Neos.Fusion:Template {
                templatePath = 'resource://Acme.DemoCom/Private/Templates/Page/
↪Default.html'
                sectionName = 'bodyScripts'
        }
}
```

The order of the includes can be specified with the *@position* property inside the *Array* object. This is especially handy for including JavaScript libraries and plugins in the correct order:

```
page.head {
        jquery = '<script src="//code.jquery.com/jquery-1.10.1.min.js"></script>'

        javascripts.jquery-ui = '<script src="path-to-jquery-ui"></script>'
        javascripts.jquery-ui.@position = 'after jquery'
}
```

CSS and JavaScript restrictions in a Neos Site

Very little constraints are imposed through Neos for including JavaScripts or stylesheets. But since the Neos user interface itself is built with HTML, CSS and JavaScript itself, some caveats exist.

Since the generated markup contains no stylesheets by default and the generated JS is minimal, those restrictions affect only the display of the page to the editor when logged in to the Neos editing interface.

In this case, the Neos styles are included and a number of JavaScript libraries are loaded, among them jQuery, Ember JS and VIE. The styles are all confined to a single root selector and for JavaScript the impact is kept as low as possible through careful scoping.

CSS Requirements

- the <*body*> tag is not allowed to have a CSS style with *position:relative*, as this breaks the positions of modal dialogs we show at various places. *Zurb Foundation* is one well-known framework which sets this as default, so if you use it, then fix the error with *body { position: static }*.

TODO check if this is still true

JavaScript Requirements

TODO "what about the UI below a single DOM element idea"

Adjusting the HTTP response

It is possible to set HTTP headers and the status code of the response from Fusion. See *Neos.Fusion:Http.Message* for an example.

3.4 Content Dimensions

3.4.1 Introduction

Content dimensions are a generic concept to have multiple *variants* of a node. A dimension can be anything like "language", "country" or "customer segment". The content repository supports any number of dimensions. Node variants can have multiple values for each dimension and are connected by the same identifier. This enables a *single-tree* approach for localization, personalization or other variations of the content in a site.

If content is rendered and thus fetched from the content repository, it will always happen in a *context*. This context contains a list of values for each dimension that specifies which dimension values are visible and in which *fallback order* these should apply. So the same node variants can yield different results depending on the context that is used to fetch the nodes.

Dimension presets assign a name to the list of dimension values and are used to display dimensions in the user interface or in the routing. They represent the allowed combinations of dimension values.

Tip: See the *Translating content* cookbook for a step-by-step guide to create a multi-lingual website with Neos.

3.4.2 Dimension Configuration

The available dimensions and presets can be configured via settings:

```
Neos:
  ContentRepository:
    contentDimensions:

      # Content dimension "language" serves for translation of content into different␣
↪languages. Its value specifies
      # the language or language variant by means of a locale.
      'language':
        # The default dimension that is applied when creating nodes without␣
↪specifying a dimension
        default: 'mul_ZZ'
        # The default preset to use if no URI segment was given when resolving␣
↪languages in the router
        defaultPreset: 'all'
        label: 'Language'
        icon: 'icon-language'
        presets:
          'all':
            label: 'All languages'
```

(continues on next page)

(continued from previous page)

```
            values: ['mul_ZZ']
            uriSegment: 'all'
          # Example for additional languages:

          'en_GB':
            label: 'English (Great Britain)'
            values: ['en_GB', 'en_ZZ', 'mul_ZZ']
            uriSegment: 'gb'
          'de':
            label: 'German (Germany)'
            values: ['de_DE', 'de_ZZ', 'mul_ZZ']
            uriSegment: 'de'
```

The Neos ContentRepository and Neos packages don't provide any dimension configuration per default.

3.4.3 Preset Constraints

Neos can be configured to work with more than one content dimension. A typical use case is to define separate dimensions for language and country: pages with product descriptions may be available in English and German, but the English content needs to be different for the markets target to the UK or Germany respectively. However, not all possible combinations of `language` and `country` make sense and thus should not be accessible. The allowed combinations of content dimension presets can be controlled via the preset constraints feature.

Consider a website which has dedicated content for the US, Germany and France. The content for each country is available in English and their respective local language. The following configuration would make sure that the combinations "German – US", "German - France", "French - US" and "French - Germany" are not allowed:

```
Neos:
  ContentRepository:
    contentDimensions:
      'language':
        default: 'en'
        defaultPreset: 'en'
        label: 'Language'
        icon: 'icon-language'
        presets:
          'en':
            label: 'English'
            values: ['en']
            uriSegment: 'en'
          'de':
            label: 'German'
            values: ['de']
            uriSegment: 'de'
            constraints:
              country:
                'us': false
                'fr': false
          'fr':
            label: 'French'
            values: ['fr']
            uriSegment: 'fr'
            constraints:
              country:
                'us': false
```

(continues on next page)

(continued from previous page)

```
            'de': false
      'country':
        default: 'us'
        defaultPreset: 'us'
        label: 'Country'
        icon: 'icon-globe'
        presets:
          'us':
            label: 'United States'
            values: ['us']
            uriSegment: 'us'
          'de':
            label: 'Germany'
            values: ['de']
            uriSegment: 'de'
          'fr':
            label: 'France'
            values: ['fr']
            uriSegment: 'fr'
```

Instead of configuring every constraint preset explicitly, it is also possible to allow or disallow all presets of a given dimension by using the wildcard identifier. The following configuration has the same effect like in the previous example:

```
Neos:
  ContentRepository:
    contentDimensions:
      'language':
        default: 'en'
        defaultPreset: 'en'
        label: 'Language'
        icon: 'icon-language'
        presets:
          'en':
            label: 'English'
            values: ['en']
            uriSegment: 'en'
          'de':
            label: 'German'
            values: ['de']
            uriSegment: 'de'
            constraints:
              country:
                'de': true
                '*': false
          'fr':
            label: 'French'
            values: ['fr']
            uriSegment: 'fr'
            constraints:
              country:
                'fr': true
                '*': false
      'country':
        default: 'us'
        defaultPreset: 'us'
```

(continues on next page)

(continued from previous page)

```
        label: 'Country'
        icon: 'icon-globe'
        presets:
          'us':
            label: 'United States'
            values: ['us']
            uriSegment: 'us'
          'de':
            label: 'Germany'
            values: ['de']
            uriSegment: 'de'
          'fr':
            label: 'France'
            values: ['fr']
            uriSegment: 'fr'
```

While the examples only defined constraints in the `language` dimension configuration, it is perfectly possible to additionally or exclusively define constraints in `country` or other dimensions.

3.4.4 Migration of existing content

Adjusting content dimensions configuration can lead to issues for existing content. When a new content dimension is added, a corresponding value needs to be added to existing content, otherwise no nodes would be found.

This can be done with a node migration which is included in the `Neos.ContentRepository` package:

```
./flow node:migrate 20150716212459
```

This migration adds missing content dimensions by setting the default value on all existing nodes, if not already set.

Alternatively a custom node migration can be created allowing flexibility and constraints. See *Node Migration Reference*.

3.4.5 Routing

Neos provides a route-part handler that will include a prefix with the value of the `uriSegment` setting of a dimension preset for all configured dimensions. This means URIs will not contain any prefix by default as long as no content dimension is configured. Multiple dimensions are joined with a _ character, so the `uriSegment` value must not include an underscore.

The default preset can have an empty *uriSegment* value. The following example will lead to URLs that do not contain *en* if the *en_US* preset is active, but will show the *uriSegment* for other languages that are defined as well:

```
Neos:
  ContentRepository:
    contentDimensions:

      'language':
        default: 'en'
        defaultPreset: 'en_US'
        label: 'Language'
        icon: 'icon-language'
        presets:
          'en':
```

(continues on next page)

(continued from previous page)

```
            label: 'English (US)'
            values: ['en_US']
            uriSegment: ''
```

The only limitation is that all segments must be unique across all dimensions. If you need non-unique segments, you can switch support for non-empty dimensions off:

```
Neos:
  Neos:
    routing:
      supportEmptySegmentForDimensions: FALSE
```

3.4.6 Limitations

In Neos 1.2 node variants can only be created by having a common fallback value in the presets. This means a node can only be translated to some other dimension value if it "shined" through from a fallback value.

In Neos 2.0, it is possible to create node variants across dimension borders, i.e. to translate an English version of a Document to German, without having fall-backs from German to English or vice versa.

Note: This is a documentation stub.

3.5 Multi Site Support

3.5.1 Separating Assets Between Sites

In multi-site setups it can become a use case to having to separate assets to a between sites. For this Neos supports creating asset collections. An asset collection can contain multiple assets, and an asset can belong to multiple collections. Additionally tags can belong to one or multiple collections.

Every site can (in the site management module) be configured to have a default asset collection. This means that when assets are uploaded in the inspector they will automatically be added to the sites collection if one is configured. When the editor opens the media browser/module it will automatically select the current sites collection.

The media browser/module allows administrators to create/edit/delete collections and also select which tags are included in a collection.

3.6 Content Cache

3.6.1 Introduction

The frontend rendering of a document node in Neos can involve many queries and operations. Doing this for every request would be too slow to achieve a feasible response time. The content cache is a feature of Fusion and supports a configurable and nested cache that can answer many requests directly from the cache without expensive operations. It is based on the Flow caching framework that supports many different cache backends, expiration and tagging.

Each Fusion path (of type object) can have its own cache configuration. These cache configurations can be nested to re-use parts of the content and have multiple cache entries with different properties on the same page. This could be

a menu or section that is the same for many pages. The nesting support is also allows to have uncached content like plugins inside cached content.

The content cache is active even when you are in editing mode. Cache entries will be *flushed* automatically whenever data has changed through a tag based strategy or when relevant files changed during development (code, templates or configuration).

Note: In Neos, you don't a have a button to clear the cache. Cache invalidation is handled by the core and can be configured to be application specific. It's really important to configure the cache correctly to avoid problems with cache invalidation.

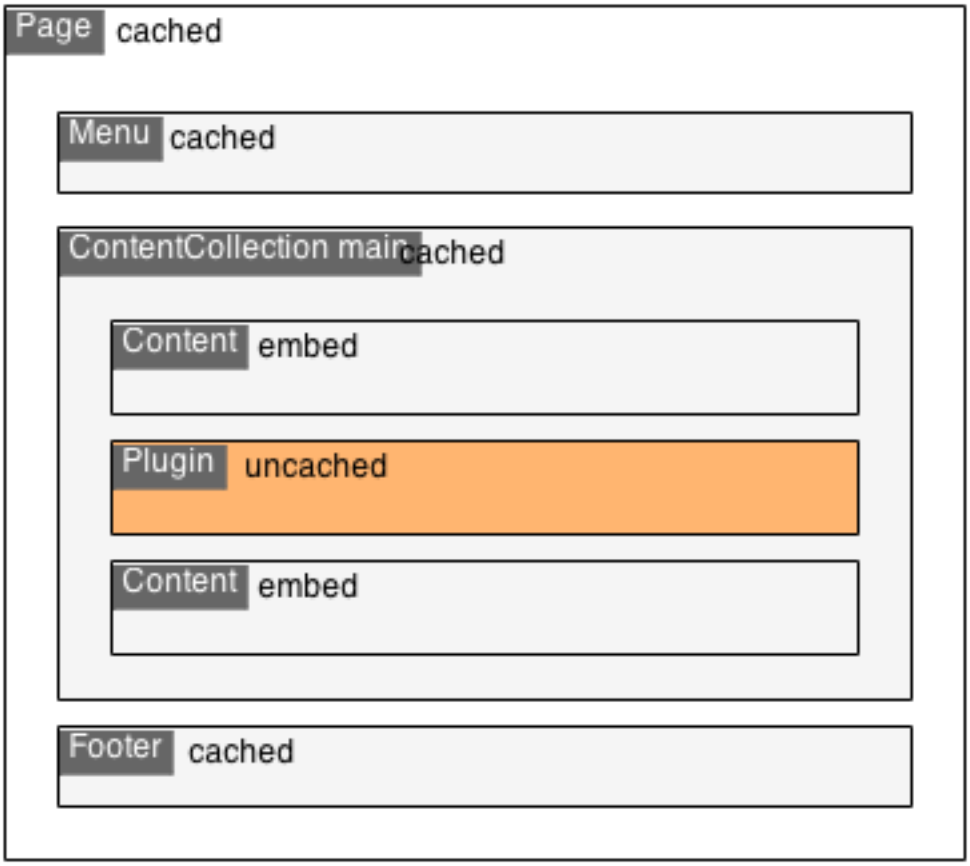


Fig. 1: An example cache hierarchy with different modes

Let's see how the content cache can help you to deliver a faster user experience.

3.6.2 The basics

The main Fusion path is `root`, you can find it in the file `Fusion/DefaultFusion.fusion` in the `Neos.Neos` package. Here is a small part of this file that shows the outermost cache configuration of the `root` path:

```
root = Neos.Fusion:Case {
        default {
                @position = 'end 9999'
                condition = TRUE
                renderPath = '/page'
        }

        @cache {
                mode = 'cached'

                maximumLifetime = '86400'

                entryIdentifier {
                        node = ${node}
                        editPreviewMode = ${node.context.currentRenderingMode.name}
                }

                entryTags {
                        # Whenever the node changes the matched condition could change
                        1 = ${Neos.Caching.nodeTag(documentNode)}
                        # Whenever one of the parent nodes changes the layout could␣
↪change
                        2 = ${Neos.Caching.descendantOfTag(documentNode)}
                }
        }
}
```

The given configuration will cache the entire page content with a unique identifier defined by the current node (the document node), the preview mode and *globally configured* entry identifiers.

Note: All `entryIdentifier` values will be evaluated and combined to a single string value (the keys will be part of the identifier and sorted alphabetically).

In the `@cache` meta property the following subproperties are allowed:

`mode` Sets the caching mode of the current path. Possible values are `'embed'` (default), `'cached'`, `'dynamic'` or `'uncached'`. Only simple string values are supported for this property.

It defaults to mode `embed` which will not create a new cache entry but store the content into the next outer `cached` entry. With mode `cached` a separate cache entry will be created for the path. Mode `uncached` can be used to always evaluate a path even if is contained inside a cached path. The `dynamic` mode evalutes a so called "discriminator" on every request and caches results differently depending on it's value. Dynamic cache mode is therefore much faster than `uncached` but slightly slower compared to `cached` mode. It is useful in situations where arguments (eg. from the request) lead to different rendering results. The `context` property should be set to configure the Fusion context variables that will be available when evaluating the uncached path.

`maximumLifetime` Set the maximum lifetime for the nearest cached path. Possible values are `null` (default), `0` (unlimited lifetime) or the amount of seconds as an integer.

If this property is declared on a path with caching mode `cached` or `dynamic` it will set the lifetime of the cache entry to the minimum of all nested `maximumLifetime` configurations (in paths with mode `embed`) and the `maximumLifetime` of the current configuration.

entryIdentifier Configure the cache entry identifier for mode `cached` or `dynamic` based on an array of values.

The prototype `Neos.Fusion:GlobalCacheIdentifiers` will be used as the base object, so global values that influence *all* cache entries can be added to that prototype, see *Global cache entry identifiers* for more details.

If this property is not set, the identifier is built from all Fusion context values that are simple values or implement `CacheAwareInterface`.

The identifier string value will be a hash built over all array values including and sorted by their key.

Note: It is very important to add all values that influence the output of the current path to the `entryIdentifier` array since cache entries will be re-used across rendered documents if the same identifier is requested. In the cache hierarchy the outermost cache entry determines all the nested entries, so it's important to add values that influence the rendering for every cached path along the hierarchy.

---

entryTags Configure a set of tags that will be assigned to the cache entry for mode `cached` or `dynamic` as an array.

The correct entry tags are important to achieve an automatic flushing of affected cache entries if a node or other data in Neos was changed during editing, publishing or other actions. A number of tags with a specific pattern are flushed by default in Neos whenever a node is changed, published or discarded. See *Cache Entry Tags* for a full list.

context Configure a list of variable names that will be stored from the Fusion context for later rendering of a path with mode `uncached` or `dynamic`. Only values that are configured here will be available in Fusion when the path is evaluated in subsequent request.

Example from `Plugin.fusion`:

```
prototype(Neos.Neos:Plugin) {
        @cache {
                mode = 'uncached'
                context {
                        1 = 'node'
                        2 = 'documentNode'
                }
        }
}
```

entryDiscriminator Configure an expression that uniquely discriminates different entries of a `dynamic` cached area. The expression or Fusion object must evaluate to a string to be used as discriminator and should be different for every cache entry you want to create for this `dynamic` cached area.

Example for a `dynamic` configuration with `entryDiscriminator`:

```
prototype(Neos.Neos:Plugin) {
        @cache {
                mode = 'dynamic'
                entryIdentifier {
                  node = ${node}
                }
                entryDiscriminator = ${request.arguments.pagination}
                context {
                        1 = 'node'
                        2 = 'documentNode'
                }
```

(continues on next page)

(continued from previous page)

```
                entryTags {
                        1 = ${Neos.Caching.nodeTag(node)}
                }
        }
}
```

When using `dynamic` as the cache mode, the cache can be *disabled* by setting the `entryDiscriminator` to `false`. This can be used to make the cache behavior dependable on some context, i.e. the current request method:

```
prototype(Neos.NodeTypes:Form) {
        @cache {
                mode = 'dynamic'
                entryIdentifier {
                  node = ${node}
                }
                entryDiscriminator = ${request.httpRequest.methodSafe ? 'static' :␣
↪false}
                context {
                        1 = 'node'
                        2 = 'documentNode'
                }
        }
}
```

In this example the Form will be `cached` unless the request method is unsafe (for example `POST`) in which case it is switched to `uncached`.

Cache Entry Tags

Neos will automatically flush a set of tags whenever nodes are created, changed, published or discarded. The exact set of tags depends on the node hierarchy and node type of the changed node. You should assign tags that matches one of these patterns in your configuration. You can use an Eel expression to build the pattern depending on any context variable including the node identifier or type.

To create the correct tags for your node it is important to make use if our CachingHelper. The following methods are provided by default to create all kind of patterns of tags wich will be flushed by Neos:

`Everything` Flushes cache entries for every changed node.

`${Neos.Caching.nodeTypeTag('[My.Package:NodeTypeName]', node)}` Flushes cache entries if any node with the given node type changes. `[My.Package:NodeTypeName]` needs to be replaced by any node type name. Inheritance will be taken into account, so for a changed node of type `Neos.NodeTypes:Page` the tags `NodeType_Neos.NodeTypes:Page` and `NodeType_Neos.Neos:Document` (and some more) will be flushed. The second property node is needed to calculate the correct context to create tags for. Notice: In earlier versions of Neos we just used a plain String `NodeType_[My.Package:NodeTypeName]` which could lead into unwanted cache flush behaviours

`${Neos.Caching.nodeTag(node)}` Flushes cache entries if the node changes. Notice: In earlier versions of Neos we just used a plain String `Node_[Identifier]` which could lead into unwanted cache flush behaviours. `Identifier` had to be replaced by a valid node identifier. You might want to use `${Neos.Caching.nodeTagForIdentifier("identifier")}` if you don't have a node instance but only a node identifier as string.

`${Neos.Caching.descendantOfTag(node)}` Flushes cache entries if a child node of the node changes. Notice: In earlier versions of Neos we just used a plain String `DescendantOf_[Identifier]` which could lead into unwanted cache flush behaviours. `Identifier` had to be replaced by a valid node identifier.

Example:

```
prototype(Neos.Neos:ContentCollection) {
        #...

        @cache {
                #...

                entryTags {
                        1 = ${Neos.Caching.nodeTag(node)}
                        2 = ${Neos.Caching.descendantOfTag(contentCollectionNode)}
                }
        }
}
```

The `ContentCollection` cache configuration declares tags that will flush the cache entry for the collection if any of it's descendants (direct or indirect child) changes. So editing a node inside the collection will flush the whole collection cache entry and cause it to re-render.

Note: When using `cached` as the cache mode, your `entryTags` should always contain the node identifier. Otherwise, the cache will not be flushed when you make changes to the node itself, which will lead to unexpected behavior in the Neos backend:

```
@cache {
        mode = 'cached'
        entryTags {
                1 = ${Neos.Caching.nodeTag(node)}
                2 = ... additional entry tags ...
        }
}


Notice: In earlier versions of Neos there might be some problems with unwanted cache␣
↪flush behaviours. To make sure
to avoid this always use the CachingEelHelper. See https://github.com/neos/neos-
↪development-collection/issues/2096
for more informations
```

3.6.3 Default cache configuration

The following list of Fusion prototypes is cached by default:

- Neos.Neos:Breadcrumb
- Neos.Neos:Menu
- Neos.Neos:Page
- Neos.Neos:ContentCollection (see note)

The following list of Fusion prototypes is uncached by default:

- Neos.NodeTypes:Form
- Neos.Neos:Plugin

Note: The `Neos.Neos:ContentCollection` prototype is cached by default and has a cache configuration with proper identifier, tags and maximumLifetime defined. For all `ContentCollection` objects inside a `Content` object the mode is set to `embed`. This means that node types that have a `ContentCollection` do not generate a separate cache entry but are embedded in the outer *static* `ContentCollection`.

Overriding default cache configuration

You can override default cache configuration in your Fusion:

```
prototype(Neos.Neos:PrimaryContent).@cache.mode = 'uncached'
```

You can also override cache configuration for a specific Fusion Path:

```
page.body.content.main {
    prototype(Neos.Neos:Plugin).@cache.mode = 'cached'
}
```

3.6.4 Global cache entry identifiers

Information like the request format or base URI that was used to render a site might have impact on all generated URIs. Depending on the site or application other data might influence the uniqueness of cache entries. If an `entryIdentifier` for a cached path is declared without an object type, it will default to `Neos.Fusion:GlobalCacheIdentifiers`:

```
prototype(My.Package:ExampleNode) {
        @cache {
                mode = 'cached'

                # This is the default if no object type is specified
                # entryIdentifier = Neos.Fusion:GlobalCacheIdentifiers
                entryIdentifier {
                        someValue = ${q(node).property('someValue')}
                }
        }
}
```

This prototype can be extended to add or remove custom global values that influence *all* cache entries without a specific object type:

```
prototype(Neos.Fusion:GlobalCacheIdentifiers) {
        myRequestArgument = ${request.arguments.myArgument}
}
```

You can use a `Neos.Fusion:RawArray` to explicitly specify the values that are used for the entry identifier:

```
prototype(My.Package:ExampleNode) {
        @cache {
                mode = 'cached'

                entryIdentifier = Neos.Fusion:RawArray {
                        someValue = ${q(node).property('someValue')}
                }
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Security Context

In addition to entry identifiers configured in Fusion, the Security Context Hash is added to the identifier of all cached segments. This hash is build from the roles of all authenticated accounts and cache identifiers from custom global objects (exposed through Neos.Flow.aop.globalObjects) implementing CacheAwareInterface.[1]

3.6.5 Tuning your cache

Change the cache backend

By default, all cache entries are stored on the local filesystem. You can change this in `Caches.yaml`, the example below will use the Redis backend for the content cache:

```
Neos_Fusion_Content:
  backend: Neos\Cache\Backend\RedisBackend
```

Note: The best practice is to change the cache configuration in your distribution.

3.7 Permissions & Access Management

3.7.1 Introduction

A common requirement, especially for larger websites with many editors, is the possibility to selectively control access to certain backend tools and parts of the content. For example so that editors can only edit certain pages or content types or that they are limited to specific workspaces. These access restrictions are used to enforce certain workflows and to reduce complexity for editors.

Neos provides a way to define Access Control Lists (ACL) in a very fine-grained manner, enabling the following use cases:

- hide parts of the node tree completely (useful for multi-site websites and frontend-login)
- show only specific Backend Modules
- allow to create/edit only specific Node Types
- allow to only edit parts of the Node Tree
- allow to only edit a specific dimension

The underlying security features of Flow provide the following generic possibilities in addition:

- protect arbitrary method calls
- define the visibility of arbitrary elements depending on the authenticated user

[1] Custom Global Objects are explained in detail in the Flow documentation: http://flowframework.readthedocs.io/en/stable/TheDefinitiveGuide/PartIII/Security.html#content-security-entityprivilege.

Privilege targets define what is restricted, they are defined by combining privileges with matchers, to address specific parts of the node tree. A user is assigned to one or more specific roles, defining who the user is. For each role, a list of privileges is specified, defining the exact permissions of users assigned to each role.

In the Neos user interface, it is possible to assign a list of multiple roles to a user. This allows to define the permissions a user actually has on a fine-grained level. Additionally, the user management module has basic support for multiple accounts per user: a user may, for example, have one account for backend access and another one for access to a member-only area on the website.

As a quick example, a privilege target giving access to a specific part of the node tree looks as follows:

```
'Neos\ContentRepository\Security\Authorization\Privilege\NodeTreePrivilege':
  'YourSite:EditWebsitePart':
    matcher: 'isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")'
```

3.7.2 Adjusting and defining roles

Neos comes with a number of predefined roles that can be assigned to users:

Role	Parent role(s)	Description
Neos.ContentRepository:Administrator		A no-op role for future use
Neos.Neos:AbstractEditor	Neos.ContentRepository:Administrator	Grants the very basic things needed to use Neos at all
Neos.Neos:LivePublisher		A "helper role" to allow publishing to the live workspace
Neos.Neos:RestrictedEditor	Neos.Neos:AbstractEditor	Allows to edit content but not publish to the live workspace
Neos.Neos:Editor	Neos.Neos:AbstractEditor Neos.Neos:LivePublisher	Allows to edit and publish content
Neos.Neos:Administrator	Neos.Neos:Editor	Everything the Editor can do, plus admin things

To adjust permissions for your editors, you can of course just adjust the existing roles (*Neos.Neos:RestrictedEditor* and *Neos.Neos:Editor* in most cases). If you need different sets of permissions, you will need to define your own custom roles, though.

Those custom roles should inherit from RestrictedEditor or Editor and then grant access to the additional privilege targets you define (see below).

Here is an example for a role (limiting editing to a specific language) that shows this:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\EditNodePrivilege':
    # this privilegeTarget is defined to switch to an "allowlist" approach
    'Acme.Com:EditAllNodes':
      matcher: 'TRUE'

    'Acme.Com:EditFinnish':
      matcher: 'isInDimensionPreset("language", "fi")'

roles:
  'Neos.Neos:Editor':
    privileges:
      -
        privilegeTarget: 'Acme.Com:EditAllNodes'
```

(continues on next page)

(continued from previous page)

```
        permission: GRANT

  'Acme.Com:FinnishEditor':
    parentRoles: ['Neos.Neos:RestrictedEditor']
    privileges:
      -
        privilegeTarget: 'Acme.Com:EditFinnish'
        permission: GRANT
```

3.7.3 Node Privileges

Node privileges define what can be restricted in relation to accessing and editing nodes. In combination with matchers (see the next section) they allow to define privilege targets that can be granted or denied for specific roles.

Note: This is an excludelist by default, so the privilege won't match if one of the conditions don't match. So the example:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\CreateNodePrivilege':
    'Some.Package:SomeIdentifier':
      matcher: >-
        isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")
        && createdNodeIsOfType("Neos.NodeTypes:Text")
```

will actually only affect nodes of that type (and subtypes). All users will still be able to create other node types, unless you also add a more generic privilege target:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\CreateNodePrivilege':
    'Some.Package:SomeIdentifier':
      matcher: isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")
```

That will be abstained by default. It's the same with MethodPrivileges, but with those we abstain all actions by default (in Neos that is).

NodeTreePrivilege

A privilege that prevents matching document nodes to appear in the Navigate Component. It also prevents editing of those nodes in case the editor navigates to a node without using the Navigate Component (e.g. by entering the URL directly).

Usage example:

```
privilegeTargets:
  'Neos\Neos\Security\Authorization\Privilege\NodeTreePrivilege':
    'Some.Package:SomeIdentifier':
      matcher: 'isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")'
```

This defines a privilege that intercepts access to the specified node (and all of its child nodes) in the node tree.

EditNodePropertyPrivilege

A privilege that targets editing of node properties.

Usage example:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\
↪EditNodePropertyPrivilege':
    'Some.Package:SomeIdentifier':
      matcher: >-
        isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")
        && nodePropertyIsIn(["hidden", "name"])
```

This defines a privilege target that intercepts editing the “hidden” and “name” properties of the specified node (and all of its child nodes).

ReadNodePropertyPrivilege

A privilege that targets reading of node properties.

Usage example:

```
'Neos\ContentRepository\Security\Authorization\Privilege\Node\
↪ReadNodePropertyPrivilege':
  'Some.Package:SomeIdentifier':
    matcher: 'isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")'
```

This defines a privilege target that intercepts reading any property of the specified node (and all of its child-nodes).

RemoveNodePrivilege

A privilege that targets deletion of nodes.

Usage example:

```
privilegeTargets:
 'Neos\ContentRepository\Security\Authorization\Privilege\Node\RemoveNodePrivilege':
   'Some.Package:SomeIdentifier':
     matcher: 'isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")'
```

This defines a privilege target that intercepts deletion of the specified node (and all of its child-nodes).

CreateNodePrivilege

A privilege that targets creation of nodes.

Usage example:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\CreateNodePrivilege':
    'Some.Package:SomeIdentifier':
      matcher: >-
        isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")
        && createdNodeIsOfType("Neos.NodeTypes:Text")
```

This defines a privilege target that intercepts creation of Text nodes in the specified node (and all of its child nodes).

EditNodePrivilege

A privilege that targets editing of nodes.

Usage example:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\EditNodePrivilege':
    'Some.Package:SomeIdentifier':
      matcher: >-
        isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")
        && nodeIsOfType("Neos.NodeTypes:Text")
```

This defines a privilege target that intercepts editing of Text nodes on the specified node (and all of its child nodes).

ReadNodePrivilege

The ReadNodePrivilege is used to limit access to certain parts of the node tree:

With this configuration, the node with the identifier c1e528e2-b495-0622-e71c-f826614ef287 and all its child nodes will be hidden from the system unless explicitly granted to the current user (by assigning `SomeRole`):

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\ReadNodePrivilege':
    'Some.Package:MembersArea':
      matcher: 'isDescendantNodeOf("c1e528e2-b495-0622-e71c-f826614ef287")'

roles:
  'Some.Package:SomeRole':
    privileges:
      -
        privilegeTarget: 'Some.Package:MembersArea'
        permission: GRANT
```

3.7.4 Privilege Matchers

The privileges need to be applied to certain nodes to be useful. For this, matchers are used in the policy, written using Eel. Depending on the privilege, various methods to address nodes are available.

Note: Global objects in matcher expressions

Since the matchers are written using Eel, anything in the Eel context during evaluation is usable for matching. This is done by using the `context` keyword, followed by dotted path to the value needed. E.g. to access the personal workspace name of the currently logged in user, this can be used:

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\ReadNodePrivilege':
    'Neos.ContentRepository:Workspace':
      matcher: 'isInWorkspace("context.userInformation.personalWorkspaceName"))'
```

These global objects available under `context` (by default the current `SsecurityContext` imported as `securityContext` and the `UserService` imported as `userInformation`) are registered in the *Settings.yaml* file in section `aop.globalObjects`. That way you can add your own as well.

Position in the Node Tree

This allows to match on the position in the node tree. A node matches if it is below the given node or the node itself.

Signature: `isDescendantNodeOf(node-path-or-identifier)`

Parameters:

- `node-path-or-identifier` (string) The nodes' path or identifier

Applicable to: matchers of all node privileges

This allows to match on the position in the node tree. A node matches if it is above the given node.

Signature: `isAncestorNodeOf(node-path-or-identifier)`

Parameters:

- `node-path-or-identifier` (string) The nodes' path or identifier

Applicable to: matchers of all node privileges

This allows to match on the position in the node tree. A node matches if it is above the given node or anywhere below the node itself.

Signature: `isAncestorOrDescendantNodeOf(node-path-or-identifier)`

Parameters:

- `node-path-or-identifier` (string) The nodes' path or identifier

Applicable to: matchers of all node privileges

Note: The node path is not reliable because it changes if a node is moved. And the path is not "human-readable" in Neos because new nodes get a unique random name. Therefore it is best practice not to rely on the path but on the identifier of a node.

NodeType

Matching against the type of a node comes in two flavors. Combining both allows to limit node creation in a sophisticated way.

The first one allows to match on the type a node has:

Signature: `nodeIsOfType(nodetype-name)`

Parameters:

- `node-path-or-identifier` (stringlarray) an array of supported node type identifiers or a single node type identifier

Applicable to: matchers of all node privileges

Inheritance is taken into account, so that specific types also match if a supertype is given to this matcher.

The second one allows to match on the type of a node that is being created:

Signature: `createdNodeIsOfType(nodetype-identifier)`

Parameters:

- `nodetype-identifier` (stringlarray) an array of supported node type identifiers or a single node type identifier

Applicable to: matchers of the `CreateNodePrivilege`

This acts on the type of the node that is about to be created.

Workspace Name

This allows to match against the name of a workspace a node is in.

Signature: `isInWorkspace(workspace-names)`

Parameters:

- `workspace-names` (string|array) an array of workspace names or a single workspace name

Applicable to: matchers of all node privileges

Property Name

This allows to match against the name of a property that is going to be affected.

Signature: `nodePropertyIsIn(property-names)`

Parameters:

- `property-names` (string|array) an array of property names or a single property name

Applicable to: matchers of he `ReadNodePropertyPrivilege` and the `EditNodePropertyPrivilege`

Content Dimension

This allows to restrict editing based on the content dimension a node is in. Matches if the currently-selected preset in the passed dimension `name` is one of `presets`.

Signature: `isInDimensionPreset(name, value)`

Parameters:

- `name` (string) The content dimension name
- `presets` (string|array) The preset of the content dimension

Applicable to: matchers of all node privileges

The following example first blocks editing of nodes completely (by defining a privilege target that always matches) and then defines a privilege target matching all nodes having a value of "de" for the "language" content dimension. That target is then granted for the "Editor" role.

```
privilegeTargets:
  'Neos\ContentRepository\Security\Authorization\Privilege\Node\EditNodePrivilege':
    # This privilegeTarget must be defined, so that we switch to an "allowlist"␣
↪approach
    'Neos.Demo:EditAllNodes':
      matcher: 'TRUE'

    'Neos.Demo:EditGerman':
      matcher: 'isInDimensionPreset("language", "de")'

roles:
  'Neos.Neos:Editor':
    privileges:
```

(continues on next page)

(continued from previous page)

```
    -
      privilegeTarget: 'Neos.Demo:EditGerman'
      permission: GRANT
```

3.7.5 Asset Privileges

Asset privileges define what can be restricted in relation to accessing Assets (images, documents, videos, ...), AssetCollections and Tags.

Note: Like Node Privileges this is an excludelist by default, so the privilege won't match if one of the conditions don't match.

ReadAssetPrivilege

A privilege that prevents reading assets depending on the following Privilege Matchers:

Asset Title

This allows to match on the title of the asset.

Signature: `titleStartsWith(title-prefix)`

Parameters:

- `title-prefix` (string) Beginning of or complete title of the asset to match

Signature: `titleEndWith(title-suffix)`

Parameters:

- `title-suffix` (string) End of title of the asset to match

Signature: `titleContains(title-prefix)`

Parameters:

- `title-prefix` (string) Part of title of the asset to match

Asset Media Type

This allows to match on the media type of the asset.

Signature: `hasMediaType(media-type)`

Parameters:

- `media-type` (string) Media Type of the asset to match (for example "application/json")

Tag

This allows to match on a label the asset is tagged with.

Signature: `isTagged(tag-label-or-id)`

Parameters:

- `tag-label-or-id` (string) Label of the Tag to match (for example "confidential") or its technical identifier (UUID)

Asset Collection

This allows to match on an Asset Collection the asset belongs to.

Signature: `isInCollection(collection-title-or-id)`

Parameters:

- `collection-title-or-id` (string) Title of the Asset Collection to match (for example "confidential-documents") or its technical identifier (UUID)

Alternatively, the `isWithoutCollection` filter to match on assets that don't belong to any Asset Collection.

Signature: `isWithoutCollection()`

Usage example:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':
    'Some.Package:ReadAllPDFs':
      matcher: 'hasMediaType("application/pdf")'

    'Some.Package:ReadConfidentialPdfs':
      matcher: 'hasMediaType("application/pdf") && isTagged("confidential")'
```

ReadAssetCollectionPrivilege

A privilege that prevents reading Asset Collections depending on the following Privilege Matchers:

Collection Title

This allows to match on the title of the Asset Collection.

Signature: `isTitled(collection-title)`

Parameters:

- `collection-title` (string) Complete title of the Asset Collection to match

Usage example:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetCollectionPrivilege':
    'Some.Package:ReadSpecialAssetCollection':
      matcher: 'isTitled("some-asset-collection")'
```

Collection Identifier

This allows to match on the technical identifier (UUID) of the Asset Collection.

Signature: `hasId(collection-id)`

Parameters:

- `collection-id` (string) Technical identifier (UUID) of the Asset Collection to match

Usage example:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetCollectionPrivilege':
    'Some.Package:ReadSpecialAssetCollection':
      matcher: 'hasId("9b13346d-960a-45e6-8e93-c2929373bc90")'
```

ReadTagPrivilege

A privilege that prevents reading tags depending on the following Privilege Matchers:

Tag Label

This allows to match on the label of the tag.

Signature: `isLabeled(tag-label)`

Parameters:

- `tag-label` (string) Complete label of the tag to match

Usage example:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadTagPrivilege':
    'Some.Package:ReadConfidentialTags':
      matcher: 'isLabeled("confidential")'
```

Tag Identifier

This allows to match on the technical identifier (UUID) of the Tag.

Signature: `hasId(tag-id)`

Parameters:

- `tag-id` (string) Technical identifier (UUID) of the Tag to match

Usage example:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadTagPrivilege':
    'Some.Package:ReadConfidentialTags':
      matcher: 'hasId("961c3c03-da50-4a77-a5b4-11d2bbab7197")'
```

Note: You can find out more about the Asset Privileges in the Neos Media documentation

3.7.6 Restricting Access to Backend Modules

Restrict Module Access

The available modules are defined in the settings of Neos. Here is a shortened example containing only the relevant parts:

```
Neos:
 Neos:
   modules:
    'management':
      controller: 'Some\Management\Controller'
      submodules:
        'workspaces':
          controller: 'Some\Workspaces\Controller'
```

Along with those settings privilege targets should be defined. Those are used to hide the module links from the UI and to protect access to the modules if no access is granted.

The targets are defined as usual in the security policy, using *ModulePrivilege*. Here is a shortened example:

```
privilegeTargets:

  'Neos\Neos\Security\Authorization\Privilege\ModulePrivilege':

    'Neos.Neos:Backend.Module.Management':
      matcher: 'management'

    'Neos.Neos:Backend.Module.Management.Workspaces':
      matcher: 'management/workspaces'
```

Now those privilege targets can be used to grant/deny access for specific roles. Internally those module privileges create a *MethodPrivilege* covering all public actions of the configured module controller. Additionally more fine-grained permissions can be configured on top.

Note: If the path of a module changes the corresponding privilege target needs to be adjusted accordingly.

See chapter *Custom Backend Modules* for more examples.

Disable Modules

To completely disable modules available in the Neos UI a setting can be used:

```
Neos:
  Neos:
    modules:
      'management':
        submodules:
          'history':
            enabled: FALSE
```

3.7.7 Limitations

Except for the assignment of roles to users there is no UI for editing security related configuration. Any needed changes have to be made to the policies in `Policy.yaml`.

3.7.8 Further Reading

The privileges specific to Neos are built based on top of the Flow security features. Read the corresponding documentation.

CHAPTER

FOUR

EXTENDING NEOS

4.1 Creating a plugin

Any Flow package can be used as a plugin with a little effort. This section will guide you through a simple example. First, we will create a really basic Flow package. Second, we'll expose this Flow package as a Neos plugin.

4.1.1 Creating a Flow package

First we will create a very simple Flow package to use for integrating it as a plugin.

Note: When developing sites the need for simple plugins will often arise. And those small plugins will be very site-specific most of the time. In these cases it makes sense to create the needed code inside the site package, instead of in a separate package.

For the sake of simplicity we will create a separate package now.

If you do not have the Kickstart package installed, you must do this now:

```
cd /your/htdocs/Neos
php /path/to/composer.phar require neos/kickstarter \*
```

Now create a package with a model, so we have something to show in the plugin:

```
./flow kickstart:package Sarkosh.CdCollection
./flow kickstart:model Sarkosh.CdCollection Album title:string year:integer␣
↪description:string rating:integer
./flow kickstart:repository Sarkosh.CdCollection Album
```

Then generate a migration to create the needed DB schema:

```
./flow doctrine:migrationgenerate
```

The command will ask in which directory the migration should be stored. Select the package Sarkosh.CdCollection. Afterwards the migration can be applied:

```
./flow doctrine:migrate
```

You should now have a package with a default controller and templates created.

Configure Access Rights

To be able to call the actions of the controller you have to configure a matching set of rights. Create a *Policy.yaml* file in *Packages/Application/Sarkosh.CdCollection/Configuration/Policy.yaml* containing:

```
privilegeTargets:
  Neos\Flow\Security\Authorization\Privilege\Method\MethodPrivilege:
    'Sarkosh.CdCollection:StandardControllerActions':
      matcher: 'method(Sarkosh\CdCollection\Controller\StandardController->
↪(index)Action())'

roles:
  'Neos.Flow:Everybody':
    privileges:
      -
        privilegeTarget: 'Sarkosh.CdCollection:StandardControllerActions'
        permission: GRANT
```

Note: If you add new actions later on you will have to extend the matcher rule to look like `(index|other|third)`.

Configure Routes

To actually call the plugin via HTTP request you have to include the Flow default-routes into the *Configuration/Routes.yaml* of your whole setup (before the Neos routes):

```
##
# Flow subroutes
-
  name: 'Flow'
  uriPattern: 'flow/<FlowSubroutes>'
  defaults:
    '@format': 'html'
  subRoutes:
    FlowSubroutes:
      package: Neos.Flow
```

The frontend of your plugin can now be called via `http://neos.demo/flow/sarkosh.cdcollection`. We specifically use the `flow` prefix here to ensure that the routes of Flow do not interfere with Neos.

Note: The routing configuration will become obsolete as soon as you use the package as as Neos-Plugin as described in the following steps.

Add data

Now you can add some entries for your CD collection in the database:

```
INSERT INTO sarkosh_cdcollection_domain_model_album (
  persistence_object_identifier, title, year, description, rating
) VALUES (
  uuid(), 'Jesus Christ Superstar', '1970',
  'Jesus Christ Superstar is a rock opera by Andrew Lloyd Webber, with lyrics by Tim␣
↪Rice.',
  '5'
);
```

(or using your database tool of choice) and adjust the templates so a list of CDs is shown. When you are done with that, you can make a plugin out of that.

As an optional step you can move the generated package from its default location *Packages/Application/* to *Packages/Plugins*. This is purely a convention and at times it might be hard to tell an "application package" from a "plugin", but it helps to keep things organized. Technically it has no relevance.

```
mkdir Packages/Plugins
mv Packages/Application/Sarkosh.CdCollection Packages/Plugins/Sarkosh.CdCollection
```

If you do this, it is important to rescan the available packages:

```
./flow flow:package:rescan
```

After this, you can use the Plugin with the same url `http://neos.demo/flow/sarkosh.cdcollection`

4.1.2 Converting a Flow Package Into a Neos Plugin

To activate a Flow package as a Neos plugin, you only need to provide two configuration blocks.

Add a NodeType

First, you need to add a new *node type* for the plugin, such that the user can choose the plugin from the list of content elements:

Add the following to *Configuration/NodeTypes.yaml* of your package:

```
'Sarkosh.CdCollection:Plugin':
  superTypes:
    'Neos.Neos:Plugin': TRUE
  ui:
    label: 'CD Collection'
    group: 'plugins'
```

This will add a new entry labeled "CD Collection" to the "Plugins" group in the content element selector (existing groups are *General*, *Structure* and *Plugins*).

Configure Fusion

Second, the rendering of the plugin needs to be specified using Fusion, so the following Fusion needs to be added to your package.

Resources/Private/Fusion/Plugin.fusion:

```
prototype(Sarkosh.CdCollection:Plugin) < prototype(Neos.Neos:Plugin)
prototype(Sarkosh.CdCollection:Plugin) {
      package = 'Sarkosh.CdCollection'
      controller = 'Standard'
      action = 'index'
}
```

Finally tweak your site package's *Root.fusion* and include the newly created Fusion file:

```
include: Plugin.fusion
```

Now log in to your Neos backend (you must remove the Flow routes again), and you will be able to add your plugin just like any other content element.

To automatically include the Root.fusion in Neos you have to add the following lines to the *Configuration/Settings.yaml* of your Package:

```
Neos:
  Neos:
    fusion:
      autoInclude:
        'Sarkosh.CdCollection': TRUE
```

Use Fusion to configure the Plugin

To hand over configuration to your plugin you can add arbitrary Fusion values to *Resources/Private/Fusion/Plugin.fusion*:

```
prototype(Sarkosh.CdCollection:Plugin) {
      ...
      myNodeName = ${q(node).property('name')}
}
```

In the controller of your plugin you can access the value from Fusion like this.

```
$myNodeName = $this->request->getInternalArgument('__myNodeName');
```

4.1.3 Linking to a Plugin

Inside of your Plugin you can use the usual `f:link.action` and `f:uri.action` ViewHelpers from fluid to link to other ControllerActions:

```
<f:link.action package="sarkosh.cdcollection" controller="standard" action="show"␣
↪arguments="{collection: collection}" />
```

If you want to create links to your plugin from outside the plugin context you have to use one of the following methods.

To create a link to a ControllerAction of your Plugin in Fusion you can use the following code:

```
link = Neos.Neos:NodeUri {
      # you have to identify the document that contains your plugin somehow
      node = ${q(site).find('[instanceof Sarkosh.CdCollection:Plugin]').first().
↪closest('[instanceof Neos.Neos:Document]').get(0)}
      absolute = true
      additionalParams = ${{'--sarkosh_cdcollection-plugin': {'@package': 'sarkosh.
↪cdcollection', '@controller':'standard', '@action': 'show', 'collection':␣
↪collection}}}
}
```

The same code in a fluid template looks like this:

```
{namespace neos=Neos\Neos\ViewHelpers}
<neos:uri.node node="{targetNode}" arguments="{'--sarkosh_cdcollection-plugin': {
↪'@package': 'sarkosh.cdcollection', '@controller':'standard', '@action': 'show',
↪'collection': collection}}" />
```

4.1.4 Configuring a plugin to show specific actions on different pages

With the simple plugin you created above, all of the actions of that plugin are executed on one specific page node. But sometimes you might want to break that up onto different pages. For this use case there is a node type called `Plugin View`. A plugin view is basically a view of a specific set of actions configured in your `NodeTypes.yaml`.

The steps to have one plugin which is rendered at multiple pages of your website is as follows:

1. Create your plugin as usual; e.g. like in the above example.
2. Insert your plugin at a specific page, just as you would do normally. This is later called the *Master View* of your plugin.
3. You need to define the parts of your plugin you lateron want to have separated in a different page. This is done in the `options.pluginViews` setting inside `NodeTypes.yaml` (see below).
4. Then, in Neos, insert a *Plugin View* instance on the other page where you want a part of the plugin to be rendered. In the inspector, you can then select the Plugin instance inside the *Master View* option, and afterwards choose the specific Plugin View you want to use.

You can update your *Configuration/NodeTypes.yaml* like this to configure which actions will be available for the `Plugin View`:

```
'Sarkosh.CdCollection:Plugin':
  superTypes:
    'Neos.Neos:Plugin': TRUE
  ui:
    label: 'CD Collection'
    group: 'plugins'
  options:
    pluginViews:
      'CollectionShow':
        label: 'Show Collection'
        controllerActions:
          'Sarkosh\CdCollection\Controller\CollectionController': ['show']
      'CollectionOverview':
        label: 'Collection Overview'
        controllerActions:
          'Sarkosh\CdCollection\Controller\CollectionController': ['overview']
```

When you insert a plugin view for a node the links in both of the nodes get rewritten automatically to link to the view or plugin, depending on the action the link points to. Insert a "Plugin View" node in your page, and then, in the inspector, configure the "Master View" (the master plugin instance) and the "Plugin View".

Fixing Plugin Output

If you reuse an existing flow-package a plugin in Neos and check the HTML of a page that includes your plugin, you will clearly see that things are not as they should be. The plugin is included using its complete HTML, including head and body tags. This of course results in an invalid document.

To improve that you can add a *Configration/Views.yaml* file to your Package that can be used to alter the used template and views based on certain conditions. The documentation for that can be found in the Flow Framework Documentation.

Optimizing the URLs

By default Neos will create pretty verbose urls for your plugin. To avoid that you have to configure a proper routing for your Package.

Plugin Request and Response

The plugin controller action is called as a child request within the parent request. Alike that, the response is also a child response of the parent and will be handed up to the parent.

> **Warning:** The documentation is not covering all aspects yet. Please have a Look at the *How To's* Section as well.

4.2 Custom Backend Modules

If you want to integrate custom backend functionality you can do so by adding a submodule to the administration or management section of the main menu. Alternatively a new top level section can be created either by adding a overview module like the the existing ones or a normal module.

Some possible use cases would be the integrating of external web services, triggering of import or export actions or creating of editing interfaces for domain models from other packages.

> **Warning:** This is not public API yet due to it's unpolished state and is subject to change in the future.

4.2.1 Controller Class

Implementing a Backend Module starts by creating an action controller class derived from `\Neos\Flow\Mvc\Controller\ActionController`

`Classes/Vendor/Site/Domain/Controller/BackendController`:

```
namespace Vendor\Site\Controller;

use Neos\Flow\Annotations as Flow;

```

(continues on next page)

(continued from previous page)

```
class BackendController extends \Neos\Flow\Mvc\Controller\ActionController {
        public function indexAction() {
                $this->view->assign('exampleValue', 'Hello World');
        }
}
```

4.2.2 Fluid Template

The user interface of the module is defined in a fluid template in the same way the frontend of a website is defined.

`Resources/Private/Templates/Backend/Index.html`:

```
{namespace neos=Neos\Neos\ViewHelpers}
<div class="neos-content neos-container-fluid">
        <h1></h1>
        <p>{exampleValue}</p>
</div>
```

Note: Neos comes with some ViewHelpers for easing backend tasks. Have a look at the `neos:backend` ViewHelpers from the *Neos ViewHelper Reference*

4.2.3 Configuration

To show up in the management or the administration section the module is defined in the package settings.

`Configuration/Settings.yaml`:

```
Neos:
  Neos:
    modules:
      'management':
        submodules:
          'exampleModule':
            label: 'Example Module'
            controller: 'Vendor\Site\Controller\BackendController'
            description: 'An Example for implementing Backend Modules'
            icon: 'icon-star'
```

4.2.4 Access Rights

To use the module the editors have to be granted access to the controller actions of the module.

`Configuration/Policy.yaml`:

```
privilegeTargets:

  'Neos\Neos\Security\Authorization\Privilege\ModulePrivilege':

    'Vendor.Site:BackendModule':
      matcher: 'management/exampleModule'
```

(continues on next page)

(continued from previous page)

```
roles:

  'Neos.Neos:Editor':
    privileges:
      -
        privilegeTarget: 'Vendor.Site:BackendModule'
        permission: GRANT
```

Tip: Neos contains several backend modules built with the same API which can be used for inspiration.

4.3 Custom Edit/Preview-Modes

From the beginning the Neos backend was designed to be extensible with different rendering modes users can switch depending on their use-case. In-place editing and the raw-content-editing-mode are only a small glimpse of what is possible.

It is encouraged to add custom edit- or preview modes. Use-cases could be the preview of the content in search engines or on mobile devices.

4.3.1 Add a custom Preview Mode

Edit/preview modes are added to the Neos-Backend via *Settings.yaml*.

```
Neos:
  Neos:
    userInterface:
      editPreviewModes:
        print:
          title: 'Print'
          # show as edit mode
          isEditingMode: FALSE
          # show as preview mode
          isPreviewMode: TRUE
          # render path
          fusionRenderingPath: 'print'
          # show after the existing modes
          position: 200
          # sets the width of the iframe (React UI only)
          width: 800
          # sets the height of the iframe (React UI only)
          height: 600
          # custom background color for content canvas
          backgroundColor: '#ffffff'
```

The settings `isEditingMode` and `isPreviewMode` are controlling whether the mode will show up in the section "Edit" or "Preview" of the Neos-Backend. The major difference between both sections is that inside "Preview" section the inline editing options are not activated.

The actual rendering of the edit/preview mode is configured in Fusion:

```
print < page
print {
        head {
                stylesheets.printCss = Neos.Fusion:Tag {
                        @position = 'end 10'
                        tagName = 'link'
                        attributes {
                                media = 'all'
                                rel = 'stylesheet'
                                href = Neos.Fusion:ResourceUri {
                                        path = 'resource://Neos.Demo/Public/Styles/
↪Print.css'
                                }
                        }
                }
        }
}
```

In this example the default rendering as defined in the path `page` is used and altered to include the Print.css for all media.

4.3.2 Add a custom Editing Mode

To add an editing mode instead of a preview mode the configuration in *Settings.yaml* has to be changed.

```
Neos:
  Neos:
    userInterface:
      editPreviewModes:
        print:
          isEditingMode: TRUE
          isPreviewMode: FALSE
```

Warning: It is currently possible to configure an edit/preview-mode for editing and preview at the same time. We are still unsure whether this is a bug or a feature – so this behavior may change in future releases.

4.4 Custom Editors

Note: For documentation on how to create inspector editors for the legacy Ember version of the user interface, refer to the older versions of the documentation.

Every dataType has its default editor set, which can have options applied like:

```
Neos:
  Neos:
    userInterface:
      inspector:
        dataTypes:
          'string':
            editor: 'Neos.Neos/Inspector/Editors/TextFieldEditor'
```

(continues on next page)

(continued from previous page)

```
            editorOptions:
              placeholder: 'This is a placeholder'
```

On a property level this can be overridden like:

```
Neos:
  Neos:
    userInterface:
      inspector:
        properties:
          'string':
            editor: 'My.Package/Inspector/Editors/TextFieldEditor'
            editorOptions:
              placeholder: 'This is my custom placeholder'
```

In order to implement a custom inspector editor one has to use the UI extensibility layer exposed through the *@neos-project/neos-ui-extensibility* package. See *Neos User Interface Extensibility API* for the detailed information on the topic.

Let's create a simple colour picker editor. For this, create a folder structure in your package to lok like this:

```
AcmeCom.Neos.Colorpicker
├── Configuration
│   └── Settings.yaml
├── Resources
│   ├── Private
│   │   └── Scripts
│   │       └── ColorPickerEditor
│   │           ├── package.json
│   │           ├── src
│   │           │   ├── ColorPickerEditor.js
│   │           │   ├── index.js
│   │           │   └── manifest.js
│   └── Public
└── composer.json
```

You need to have a Composer manifest (*composer.json*) in place, otherwise the package will not be picked up by Flow and loading the editor will fail:

```
{
  "name": "acmecom/neos-colorpicker",
  "type": "neos-package",
  "require": {
      "neos/neos-ui": "^1.3"
  },
  "extra": {
      "neos": {
          "package-key": "AcmeCom.Neos.Colorpicker"
      }
  }
}
```

Use the following *package.json* file:

```
{
  "scripts": {
    "build": "neos-react-scripts build",
```

(continues on next page)

(continued from previous page)

```
    "watch": "neos-react-scripts watch"
  },
  "neos": {
    "buildTargetDirectory": "../../../Public/ColorPickerEditor"
  },
  "devDependencies": {
    "@neos-project/neos-ui-extensibility": "^1.3"
  },
  "dependencies": {
    "react-color": "^2.11.1"
  }
}
```

This will put the compiled *Plugin.js* asset into the *Public/ColorPickerEditor* folder. This file has to be loaded into the host UI to be useable. Put the following configuration into *Settings.yaml* to do it:

```
Neos:
  Neos:
    Ui:
      resources:
        javascript:
          'AcmeCom.Neos.ColorPicker:ColorPickerEditor':
            resource: resource://AcmeCom.Neos.ColorPicker/Public/ColorPickerEditor/
↪Plugin.js
```

The key below *javascript* has no significance, but it is best practice to use the full package key and editor name, to avoid name clashes.

Now it is time to write the actual source code of the editor. From *index.js* we just require the *manifest.js* file:

```
require('./manifest');
```

In *manifest.js* we use the *manifest* API to get access to the *globalRegistry*, then we get the *editors* registry out of it and register our custom editor into it:

```
import manifest from '@neos-project/neos-ui-extensibility';
import ColorPickerEditor from './ColorPickerEditor';

manifest('AcmeCom.Neos.ColorPicker:ColorPickerEditor', {}, globalRegistry => {
  const editorsRegistry = globalRegistry.get('inspector').get('editors');
  editorsRegistry.set('AcmeCom.Neos.ColorPicker/ColorPickerEditor', {
    component: ColorPickerEditor
  });
});
```

And finally the editor component itself in *ColorPickerEditor.js*:

```
import React, {PureComponent} from 'react';
import PropTypes from 'prop-types';
import {SketchPicker} from 'react-color';

export default class ColorPickerEditor extends PureComponent {
  static propTypes = {
    value: PropTypes.string,
    commit: PropTypes.func.isRequired,
  };
  handleChangeColor = newColor => {
```

(continues on next page)

(continued from previous page)

```
    this.props.commit(newColor.hex);
  };
  render() {
    return <SketchPicker color={this.props.value} onChange={this.handleChangeColor}/>;
  }
}
```

Each editor component gets a few API props passed, including the current value of the editor and the *commit* callback which the editor should use to commit the new value.

That is it! Now it is time to build and use our brand new editor! To build the editor you need to run the following commands:

```
cd Resources/Private/Scripts/ColorPickerEditor
yarn
yarn build # or yarn watch
```

The first call to *yarn* will install the needed dependencies, the second call to *yarn build* actually builds the editor. During development you can use *yarn watch* to run the build process whenever the code changes.

Then include the editor for some property in a node type:

```
'Neos.NodeTypes:TextMixin':
  properties:
    color:
      ui:
        label: 'Color picker'
        inspector:
          editor: 'AcmeCom.Neos.ColorPicker/ColorPickerEditor'
```

Note: You should exclude *Resources/Private/Scripts/YamlEditor/node_modules* from version control...

4.5 Custom Eel Helper

Eel Helpers provide methods that can be used inside of Eel expressions. That is mostly used to extend the capabilities for data-aquisition and processing of Fusion.

The first step is to create the EelHelper class. Every Helper has to implement the interface `Neos\Eel\ProtectedContextAwareInterface`.

```
namespace Vendor\Site\Eel\Helper;

use Neos\Flow\Annotations as Flow;
use Neos\Eel\ProtectedContextAwareInterface;

class ExampleHelper implements ProtectedContextAwareInterface {

        /**
         * Wrap the incoming string in curly brackets
         *
         * @param $text string
         * @return string
         */
```

(continues on next page)

(continued from previous page)

```
        public function wrapInCurlyBrackets($text) {
                return '{' . $text . '}';
        }

        /**
         * All methods are considered safe, i.e. can be executed from within Eel
         *
         * @param string $methodName
         * @return boolean
         */
        public function allowsCallOfMethod($methodName) {
                return TRUE;
        }

}
```

Afterwards the namespace of the Helper has to be registered for usage in Fusion in the *Settings.yaml* of the package:

```
Neos:
  Fusion:
    defaultContext:
      'Vendor.Example': 'Vendor\Site\Eel\Helper\ExampleHelper'
```

In Fusion you can call the methods of the helper inside of EelExpressions:

```
exampleEelValue = ${Vendor.Example.wrapInCurlyBrackets('Hello World')}
```

4.6 Custom FlowQuery Operations

The FlowQuery EelHelper provides you with methods to traverse the ContentRepository. Implementing custom operations allows the creation of filters, sorting algorithms and much more.

Warning: This has not been declared a public api yet and still might change a bit in future release. Nevertheless it is an important functionality and this or a similar mechanism will still be available in the future.

4.6.1 Create FlowQuery Operation

Implementing a custom operation is done by extending the `Neos\Eel\FlowQuery\Operations\AbstractOperation` class. The Operation is implemented in the evaluate method of that class.

To identify the operation lateron in Fusion the static class variable `$shortName` has to be set.

If you pass arguments to the FlowQuery Operation they end up in the numerical array `$arguments` that is handed over to the evaluate method.

```
namespace Vendor\Site\FlowQuery\Operation;

use Neos\Eel\FlowQuery\FlowQuery;
use Neos\Eel\FlowQuery\Operations\AbstractOperation;

class RandomElementOperation extends AbstractOperation {
```

(continues on next page)

(continued from previous page)

```
        /**
         * {@inheritdoc}
         *
         * @var string
         */
        static protected $shortName = 'randomElement';

        /**
         * {@inheritdoc}
         *
         * @param FlowQuery $flowQuery the FlowQuery object
         * @param array $arguments the arguments for this operation
         * @return void
         */
        public function evaluate(FlowQuery $flowQuery, array $arguments) {
                $context = $flowQuery->getContext();
                $randomKey = array_rand($context);
                $result = array($context[$randomKey]);
                $flowQuery->setContext($result);
        }
}
```

In Fusion you can use this operation to find a random element of the main ContentCollection of the Site-Node:

```
randomStartpageContent = ${q(site).children('main').children().randomElement()}
```

Note: For overriding existing operations another operation with the same shortName but a higher priority can be implemented.

4.6.2 Create Final FlowQuery Operations

If a FlowQuery operation does return a value instead of modifying the FlowQuery Context it has to be declared `$final`.

```
namespace Vendor\Site\FlowQuery\Operation;

use Neos\Eel\FlowQuery\FlowQuery;
use Neos\Eel\FlowQuery\Operations\AbstractOperation;

class DebugOperation extends AbstractOperation {

        /**
         * If TRUE, the operation is final, i.e. directly executed.
         *
         * @var boolean
         * @api
         */
        static protected $final = TRUE;

        /**
         * {@inheritdoc}
         *
```

(continues on next page)

(continued from previous page)

```
         * @param FlowQuery $flowQuery the FlowQuery object
         * @param array $arguments the arguments for this operation
         * @return void
         */
        public function evaluate(FlowQuery $flowQuery, array $arguments) {
                return \Neos\Flow\var_dump($flowQuery->getContext(), NULL, TRUE);
        }
}
```

4.6.3 Further Reading

1. For checking that the operation can actually work on the current context a canEvaluate method can be implemented.
2. **You sometimes might want to use the Fizzle Filter Engine to use jQuery like selectors in the arguments of your** operation. Therefore you can apply a filter operation that is applied to the context as follows: `$flowQuery->pushOperation('filter', $arguments);`.

4.7 Custom Fusion Objects

By adding custom Fusion Objects it is possible to extend the capabilities of Fusion in a powerful and configurable way. If you need to write a way to execute PHP code during rendering, for simple methods, Eel helpers should be used. For more complex functionality where custom classes with more configuration options are needed, Fusion objects should rather be created.

As an example, you might want to create your own Fusion objects if you are enriching the data that gets passed to the template with external information from an API or if you have to convert some entities from identifier to domain objects.

In the example below, a Gravatar image tag is generated.

4.7.1 Create a Fusion Object Class

To create a custom Fusion object the `Neos\Fusion\FusionObjects\AbstractFusionObject` class is extended. The only method that needs to be implemented is `evaluate()`. To access values from Fusion the method `$this->fusionValue('__fusion_value_key__');` is used:

```
namespace Vendor\Site\Fusion;

use Neos\Flow\Annotations as Flow;
use Neos\Fusion\FusionObjects\AbstractFusionObject;

class GravatarImplementation extends AbstractFusionObject {

        /**
         * @return string
         */
        public function evaluate() {
                $emailAddress = $this->fusionValue('emailAddress');
                $size = $this->fusionValue('size') ?: 80;
                $gravatarImageSource = 'http://www.gravatar.com/avatar/' .␣
↪md5(strtolower(trim($emailAddress))) . '?s=' . $size . '&d=mm&r=g';
```

(continues on next page)

(continued from previous page)

```
                return '<img src="' . $gravatarImageSource . '" alt="" />';
        }

}
```

To use this implementation in Fusion, you have to define a Fusion-prototype first:

```
prototype(Vendor.Site:Gravatar) {
        @class = 'Vendor\\Site\\Fusion\\GravatarImplementation'
        emailAddress = ''
        size = 80
}
```

Afterwards the prototype can be used in Fusion:

```
garavatarImage = Vendor.Site:Gravatar
garavatarImage {
        emailAddress = 'hello@neos.io'
        size = 120
}
```

4.8 Custom Validators

Note: For documentation on how to create validators for the legacy Ember version of the user interface, refer to the older versions of the documentation.

The custom validators are created similarly to custom *Custom Editors*.

Refer to *Neos User Interface Extensibility API* for detailed information on the topic.

4.9 Custom ViewHelpers

Custom ViewHelpers are the way to extend the Fluid templating engine to the needs of your project.

Note: The full documentation for writing ViewHelpers is included in the Flow documentation This documentation is a short introduction of the basic principles.

4.9.1 Create A ViewHelper Class

If you want to create a ViewHelper that you can call from your template (as a tag), you write a php class which has to inherit from `\Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper` (or its subclasses). You need to implement only one method to write a view helper:

```
namespace Vendor\Site\ViewHelpers;
class TitleViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\AbstractViewHelper {
        public function render() {
                return 'Hello World';
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Afterwards you have to register the namespace of your ViewHelper in the template before actually using it:

```
{namespace site=Vendor\Site\ViewHelpers}
<!-- tag syntax -->
<site:title />

<!-- inline syntax -->
{site:title()}
```

Note: Please look at the *Templating* documentation for an in-depth explanation of Fluid templating.

4.9.2 Declare View Helper Arguments

There exist two ways to pass arguments to a ViewHelper that can be combined:

1. Add arguments to the render-method of the ViewHelper Class:

```
namespace Vendor\Site\ViewHelpers;

class TitleViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\
↪AbstractViewHelper {
        /**
         * Render the title and apply some magic
         *
         * @param string $title the title
         * @param string $value If $key is not specified or could not be␣
↪resolved, this value is used. If this argument is not set, child nodes␣
↪will be used to render the default
         * @return string Translated label or source label / ID key
         * @throws \Neos\FluidAdaptor\Core\ViewHelper\Exception
         */
        public function render($title, $flag = FALSE) {

                # apply magic here ...

                return '<h1>' . $title . '</h1>';
        }
}
```

2. Use the registerArgument method of the AbstractViewHelper Class:

   This is especially useful if you have to define lots of arguments or create base classes for derived ViewHelpers.

```
namespace Vendor\Site\ViewHelpers;

class TitleViewHelper extends \Neos\FluidAdaptor\Core\ViewHelper\
↪AbstractViewHelper {

        /**
         * Initialize arguments
```

(continues on next page)

(continued from previous page)

```
         *
         * @return void
         */
        public function initializeArguments() {
                $this->registerArgument('title', 'string', 'The Title to␣
↪render');
                $this->registerArgument('flag', 'boolean', 'A ');
        }

        public function render() {
                $title = $this->arguments['title'];
                $flag = $this->arguments['flag'];

                # apply magic here ...

                return '<h1>' . $title . '</h1>';
        }
}
```

4.9.3 Context and Children

If your ViewHelper contains HTML code and possibly other ViewHelpers as well, the content of the ViewHelper can be rendered and used for further processing:

```
public function render($title = NULL) {
        if ($title === NULL) {
                $title = $this->renderChildren();
        }
        return '<h1>' . $title . '</h1>';
}
```

Note: It is a good practice to support passing of the main context as argument or children for flexibility an ease of use.

Sometimes your ViewHelper has to interact with other ViewHelpers insider that are rendered via `$this->renderChildren()`. To do that you can modify the context for the fluid rendering of the children. That allows keeping the scope of every ViewHelper clean and the implementation simple.

```
public function render() {
        # get the template variable container
        $templateVariableContainer = $renderingContext->
↪getTemplateVariableContainer();
        # add a variable to the context
        $templateVariableContainer->add('salutation', 'Hello World');
        # render the children, the variable salutation is available for the child␣
↪view helpers
        $result = $this->renderChildren();
        # remove the added variable again from the context
        $templateVariableContainer->remove('salutation');
        return $result;
}
```

Note: It is a considered a good practice to create a bunch of simple ViewHelpers that interact via Fluid context instead of creating complex logic inside a single ViewHelper.

4.9.4 Further reading

1. TagBased ViewHelpers - For the common case that a ViewHelper renders a single HTML-Tag as a result there is a special base class. The TagBased ViewHelper contains automatic security measures, so if you use this, the likelyhood of cross-site-scripting vulnerabilities is greatly reduced.

   To find out more about that please lookup `AbstractTagBasedViewHelper` in the Flow documentation

2. Condition ViewHelpers - To provide ViewHelpers that are doing either this or that there is a base class `AbstractConditionViewHelper`. This can be used in cases where you cannot express your condition via `<f:if condition="..." >`. To find out more about that please lookup `AbstractTagBasedViewHelper` in the Flow-Documentation.

3. Widget ViewHelpers - If a view helper needs complex controller logic, has to interact with repositories to fetch data, needs some ajax-interaction or needs a Fluid-Template for rendering, you can create a Fluid Widget. It is possible to override the Fluid Template of a Widget in another package so this also provides a way to create extensible ViewHelpers.

4.10 Customizing the Inspector

When you add a new node type, you can customize the rendering of the inspector. Based on the first node that we created in the "CreatingContentElement" cookbook, we can add some properties in the inspector.

4.10.1 Add a simple checkbox element

This first example adds a checkbox, in a dedicated inspector section, to define if we need to hide the Subheadline property.

You can just add the following configuration to your NodesType.yaml, based on the previous cookbook example:

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml)

```
'Vendor.Site:YourContentElementName':
  ui:
    inspector:
      groups:
        advanced:
          label: 'Advanced'
          icon: 'icon-fort-awesome'
          position: 2
  properties:
    hideSubheadline:
      type: boolean
      defaultValue: TRUE
      ui:
        label: 'Hide Subheadline ?'
        reloadIfChanged: TRUE
        inspector:
          group: 'advanced'
```

You can add this property to your Fusion:

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/Root.fusion)

```
prototype(Vendor.Site:YourContentElementName) < prototype(Neos.Neos:Content) {
  templatePath = 'resource://Vendor.Site/Private/Templates/FusionObjects/
↪YourContentElementName.html'
  headline = ${q(node).property('headline')}
  subheadline = ${q(node).property('subheadline')}
  hideSubheadline = ${q(node).property('hideSubheadline')}
  text = ${q(node).property('text')}
  image = ${q(node).property('image')}
}
```

And you can use it in your Fluid template:

HTML (Vendor.Site/Private/Templates/FusionObjects/YourContentElementName.html)

```
{namespace neos=Neos\Neos\ViewHelpers}
<neos:contentElement node="{node}">
  <article>
    <header>
      <h2><neos:contentElement.editable property="headline">{headline -> f:format.
↪raw()}</neos:contentElement></h2>
      <f:if condition="{hideSubheadline}">
        <f:else>
          <h3><neos:contentElement.editable property="subheadline">{subheadline ->␣
↪f:format.raw()}</neos:contentElement></h3>
        </f:else>
      </f:if>
    </header>
    ...
  </article>
</neos:contentElement>
```

4.10.2 Add a simple selectbox element

The second example is about adding a selector to change the class of the article element:

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml)

```
'Vendor.Site:YourContentElementName':
  ui:
    inspector:
      groups:
        advanced:
          label: 'Advanced'
          position: 2
          icon: 'icon-fort-awesome'
  properties:
    articleType:
      type: string
      defaultValue: ''
      ui:
        label: 'Article Type'
        reloadIfChanged: TRUE
        inspector:
```

(continues on next page)

(continued from previous page)

```
          group: 'advanced'
          editor: Content/Inspector/Editors/SelectBoxEditor
          editorOptions:
            placeholder: 'What kind of article ...'
            values:
              '':
                label: ''
              announcement:
                label: 'Announcement'
              casestudy:
                label: 'Case Study'
              event:
                label: 'Event'
```

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/Root.fusion)

```
prototype(Vendor.Site:YourContentElementName) < prototype(Neos.Fusion:Template) {
  templatePath = 'resource://Vendor.Site/Private/Templates/FusionObjects/
↪YourContentElementName.html'
  headline = ${q(node).property('headline')}
  subheadline = ${q(node).property('subheadline')}
  articleType = ${q(node).property('articleType')}
  text = ${q(node).property('text')}
  image = ${q(node).property('image')}
}
```

HTML (Vendor.Site/Private/Templates/FusionObjects/YourContentElementName.html)

```
{namespace neos=Neos\Neos\ViewHelpers}
<neos:contentElement node="{node}">
  <article{f:if(condition:articleType,then:' class="{articleType}"')}>
    ...
  </article>
</neos:contentElement>
```

4.10.3 Select multiple options in a selectbox element

For selecting more than one item with a slect box the type of the property has to be set to `array`.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Vendor.Site:YourContentElementName':
  properties:
    tags:
      type: array
      ...
      ui:
        inspector:
          ...
          editor: Content/Inspector/Editors/SelectBoxEditor
          editorOptions:
            multiple: TRUE
            allowEmpty: FALSE
            values:
              ...
```

4.10.4 Use custom DataSources for a selectbox element

To add custom selectbox-options, Neos uses *data sources* for the inspector that can be implemented in PHP. See *Data sources* for more details.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Vendor.Site:YourContentElementName':
  properties:
    articleType:
      ui:
        inspector:
          editor: Content/Inspector/Editors/SelectBoxEditor
          editorOptions:
            dataSourceIdentifier: 'acme-yourpackage-test'
```

4.10.5 Remove fields from an existing Node Type

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Neos.Neos:Plugin':
  properties:
    package:    [ ]
    subpackage: [ ]
    controller: [ ]
    action:     [ ]
```

4.10.6 Remove a selectbox option from an existing Node Type

Removing a selectbox option, can be done by simply edition your NodeTypes.yaml.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Neos.Neos:Shortcut':
  properties:
    targetMode:
      ui:
        inspector:
          editorOptions:
            values:
              parentNode: ~
```

It is also possible to add *Custom Editors* and use *Custom Validators*.

4.11 Data sources

Data sources allow easy integration of data source end points, to provide data to the editing interface without having to define routes, policies, controller.

Data sources can be used for various purposes, however the return format is restricted to JSON. An example of their usage is as a data provider for the inspector SelectBoxEditor (see *Property Type: string / array<string> SelectBoxEditor – Dropdown Select Editor* for details).

A data source is defined by an identifier and this identifier has to be unique.

To implement a data source, create a class that implements `Neos\Neos\Service\DataSource\DataSourceInterface`, preferably by extending `Neos\Neos\Service\DataSource\AbstractDataSource`. Then set the static protected property `identifier` to a string. Make sure you use a unique identifier, e.g. `acme-demo-available-dates`.

Then implement the `getData` method, with the following signature:

```
/**
 * Get data
 *
 * The return value must be JSON serializable data structure.
 *
 * @param NodeInterface $node The node that is currently edited (optional)
 * @param array $arguments Additional arguments (key / value)
 * @return mixed JSON serializable data
 * @api
 */
public function getData(NodeInterface $node = null, array $arguments);
```

The return value of the method will be JSON encoded.

Data sources are available with the following URI pattern `/neos/service/data-source/<identifier>`, which can be linked to using the follow parameters:

- `@package`: 'Neos.Neos'
- `@subpackage`: 'Service'
- `@controller`: 'DataSource'
- `@action`: 'index
- `@format`: 'json'
- `dataSourceIdentifier`: '<identifier>'

Arbitrary additional arguments are allowed. Additionally the routing only accepts `GET` requests.

If additional arguments are provided then they will automatically be available in the `$arguments` parameter of the `getData` method. Additional arguments will not be property mapped, meaning they will contain their plain value. However if an argument with the key `node` is provided, it will automatically be converted into a node. Provide a valid node path to use this, and keep in mind that the `node` argument is restricted to this use-case. This is done to make working with nodes easy.

The `dataSourceIdentifier` will automatically be removed from the `arguments` parameter.

Note: Data sources are restricted to only be accessible for users with the `Neos.Neos:Backend.DataSource` privilege, which is included in the `Neos.Neos:Editor` role. This means that a user has to have access to the backend to be able to access a data point.

Example `TestDataSource.php`:

```
<?php
namespace Acme\YourPackage\DataSource;

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

class TestDataSource extends AbstractDataSource {

```

(continues on next page)

(continued from previous page)

```
    /**
     * @var string
     */
    static protected $identifier = 'acme-yourpackage-test';

    /**
     * Get data
     *
     * @param NodeInterface $node The node that is currently edited (optional)
     * @param array $arguments Additional arguments (key / value)
     * @return array JSON serializable data
     */
    public function getData(NodeInterface $node = NULL, array $arguments)
    {
        return isset($arguments['integers']) ? array(1, 2, 3) : array('a', 'b', 'c');
    }
}
```

4.12 Interaction with the Neos backend

Note: For the list of events of the legacy Ember version of the user interface, refer to the older versions of the documentation. The events documented here exist mostly for backwards-compatibility reasons, as the current React UI provides a much more powerful extensibility layer. See *Neos User Interface Extensibility API* for the detailed information on the topic.

4.12.1 JavaScript events

Some sites will rely on JavaScript initialization when the page is rendered, typically on DocumentReady. The Neos backend will however often reload the page via Ajax whenever a node property is changed, and this might break functionality on sites relying on custom JavaScript being executed on DocumentReady.

To fix this, the Neos backend will dispatch an event when the node is added or removed from the page via ajax, and site specific JavaScript can listen on this event to trigger whatever code is needed to render the content correctly.

```
document.addEventListener('Neos.NodeCreated', function(event) {
      // Do stuff
}, false);
```

The event object given, will always have the message and time set on event.detail. Some events might have more attributes set.

The Neos backend will dispatch events that can be listened on when the following events occur:

- **Neos.NodeCreated** When a new node was added to the document. The event has a reference to the DOM element in `event.detail.element`. Additional information can be fetched through the element's attributes.
- **Neos.NodeRemoved** When a new node was removed from the document. The event has a reference to the DOM element in `event.detail.element`. Additional information can be fetched through the element's attributes.

- **Neos.NodeSelected** When a node existing on the page is selected. The event has a reference to the DOM element in `event.detail.element` and the node model object in `event.detail.node`. Additional information can be fetched through the node model.

Example of interacting with the selected node element using the `NodeSelected` event.

```
document.addEventListener('Neos.NodeSelected', function (event) {
  const node = event.detail.node;
  if (node.get('nodeType') === 'Acme:Demo') {
      console.log(node.get('properties.title'));
  }
}, false);
```

Example of listening for the `LayoutChanged` event.

4.13 Rendering special formats (CSV, JSON, XML, . . .)

Rendering an RSS feed as XML or a document in a different format than HTML is possible by configuring a new route and adding a Fusion path that renders the format.

Let's have a look at an example that introduce a `vcard` format to render an imaginary `Person` document node type.

4.13.1 Routing

`Configuration/Routes.yaml` in your site package:

```
-
        name: 'Neos :: Frontend :: Document node with vCard format'
        uriPattern: '{node}.vcf'
        defaults:
                '@package': Neos.Neos
                '@controller': Frontend\Node
                '@action': show
                '@format': vcard
        routeParts:
                node:
                        handler: Neos\Neos\Routing\
↪FrontendNodeRoutePartHandlerInterface
        appendExceedingArguments: true
```

This will register a new route to nodes with the `vcard` format. URIs with that format will get an `.vcf` extension.

Global `Configuration/Routes.yaml` (before the Neos subroutes):

```
##
# Site package subroutes

-
  name: 'MyPackage'
  uriPattern: '<MyPackageSubroutes>'
  subRoutes:
        'MyPackageSubroutes':
          package: 'My.Package'

##
```

(continues on next page)

(continued from previous page)

```
# Neos subroutes
# ...
```

This will add the new route from the site package before the Neos subroutes.

4.13.2 Fusion

The `root` case in the default Fusion will render every format that is different from `html` by rendering a path with the format value.

Root.fusion:

```
# Define a path for rendering the vcard format
vcard = Neos.Fusion:Case {
        person {
                condition = ${q(node).is('[instanceof My.Package:Person]')}
                type = 'My.Package:Person.Vcard'
        }
}

# Define a prototype to render a Person document as a vcard
prototype(My.Package:Person.Vcard) < prototype(Neos.Fusion:Http.Message) {
        # Set the Content-Type header
        httpResponseHead {
                headers.Content-Type = 'text/x-vcard;charset=utf-8'
        }
        content = My.Package:Person {
                templatePath = 'resource://My.Package/Private/Templates/NodeTypes/
↪Person.Vcard.html'
                # Set additional variables for the template
        }
}
```

4.14 Neos User Interface Extensibility API

At the heart of the Neos UI lies the system of registries – key-value stores that contain system components. The registries are populated through the *manifest* API command that is exposed through the neos-ui-extensibility package.

4.14.1 Inspector-specific Registries

Editors

Way to retrieve:

```
globalRegistry.get('inspector').get('editors')
```

Contains all inspector editors. The key is an editor name (such as *Neos.Neos/Inspector/Editors/SelectBoxEditor*), and the values are objects of the following form:

```
{
  component: TextInput // the React editor component to use. Required
  hasOwnLabel: true|false // whether the component renders the label internally or not
}
```

Component Wiring

Every component gets the following properties (see *EditorEnvelope/index.js*)

- *identifier*: an identifier which can be used for HTML ID generation
- *label*: the label
- *value*: the value to display
- *propertyName*: name of the node property to edit
- *options*: additional editor options
- *commit*: a callback function when the content changes.
  - 1st argument: the new value
  - 2nd argument (optional): an object whose keys are *saveHooks* to be triggered, the values are hook-specific options. Example: `{'Neos.UI:Hook.BeforeSave.CreateImageVariant': nextImage}`
- *renderSecondaryInspector*:
  - 1st argument: a string identifier of the second inspector; used to implement toggling of the inspector when calling this method twice.
  - 2nd argument: a callback function which can be used to render the secondary inspector. The callback function should return the secondary inspector content itself; or "undefined/null" to close the secondary inspector. Example usage: `props.renderSecondaryInspector('IMAGE_CROPPING', () => <MySecondaryInspectorContent />)`

Secondary Editors

Way to retrieve:

```
globalRegistry.get('inspector').get('editors')
```

Contains all secondary inspector editors, which can be used to provide additional, more complex functionality that needs more space of the UI than the inspector panel can provide itself.

Use it like the registry for editors.

Views

Way to retrieve:

```
globalRegistry.get('inspector').get('views')
```

Contains all inspector views.

Use it like the registry for editors.

Save Hooks

Way to retrieve:

```
globalRegistry.get('inspector').get('saveHooks')
```

Sometimes, it is needed to run code when the user presses "Apply" inside the Inspector.

Example: When the user cropped a new image, on "Apply", a new *imageVariant* must be created on the server, and then the identity of the new *imageVariant* must be stored inside the value of the image.

The process is as follows:

- When an editor wants its value to be post-processed, it calls `props.commit(newValue, {hookName: hookOptions})`
- Then, when pressing "Apply" in the UI, the hookNames are resolved inside this *saveHooks* registry.

Hook Definitions

Every entry inside this registry is a function of the following signature:

```
(valueSoFar, hookOptions) => {
  return new value; // can also return a new Promise.
}
```

4.14.2 Validators

Way to retrieve:

```
globalRegistry.get('validators')
```

Contains all server feedback handlers.

The key is the server-feedback-handler-type, and the value is a function with the following signature:

```
(feedback, store) => {
  // do whatever you like here
}
```

4.14.3 Frontend Configuration

Any settings under *Neos.Neos.Ui.frontendConfiguration* would be available here.

Might be used also for third-party packages to deliver own settings to the UI, but this is still experimental.

Settings from each package should be prefixed to avoid collisions (unprefixed settings are reserved for the core UI itself), e.g.:

```
Neos:
  Neos:
    Ui:
      frontendConfiguration:
        'Your.Own:Package':
          someKey: someValue
```

Then it may be accessed as:

```
globalRegistry.get('frontendConfiguration').get('Your.Own:Package').someKey
```

4.14.4 Inline Editors

Way to retrieve:

```
globalRegistry.get('inlineEditors')
```

Each key in this registry should be a unique identifier for an inline editor, that can be referenced in a node type configuration.

Each entry in this registry is supposed to consist of an object with the following structure:

```
{
  bootstrap: myBootstrapFunction,
  createInlineEditor: myInlineEditorFactoryFunction
}
```

bootstrap is called only once during the global initialization of the guest frame. It is not required to do anything in this function, but it is possible to prepare the guest frame environment, if any global variables must be defined or other initialization routines must be run in order for the inline editor to work.

bootstrap will receive an API Object as its first parameter, with the following methods:

- *setFormattingUnderCursor*: Will dispatch the respective action from the *@neos-project/neos-ui-redux-store* package (*actions.UI.ContentCanvas.setFormattingUnderCursor*)
- *setCurrentlyEditedPropertyName*: Will dispatch the respective action from the *@neos-project/neos-ui-redux-store* package (*actions.UI.ContentCanvas.setCurrentlyEditedPropertyName*)

createInlineEditor is called on every DOM node in the guest frame that represents an editable property. It is supposed to handle the initialization and display of an inline editor.

createInlineEditor will receive an object as its first parameter, with the following properties:

- *propertyDomNode*: The DOM node associated with the editable property
- *propertyName*: The name of the editable property
- *contextPath*: The contextPath of the associated node
- *nodeType*: The nodeType of the associated node

- *editorOptions*: The configuration for this inline editor
- *globalRegistry*: The global registry
- *persistChange*: Will dispatch the respective action from *@neos-project/neos-ui-redux-store* package (*actions.Changes.persistChanges*)

4.14.5 CKEditor5-specific registries

The integration of CKeditor5 is dead simple and tries to introduce a minimal amount of abstractions on top of CKeditor5. There are only two registries involved in configuring it: *config* and *richtextToolbar*

Configuration of CKeditor5

Way to retrieve:

```
globalRegistry.get('ckEditor5').get('config')
```

In CKE all things are configured via a single configuration object: plugins, custom configs, etc (@see https://docs.ckeditor.com/ckeditor5/latest/builds/guides/integration/configuration.html)

This registry allows to register a custom configuration processor that takes a configuration object, modifies it and returns a new one. Example:

```
config.set('doSomethingWithConfig' (ckeConfig, editorOptions) => {
  ckeConfig.mySetting = true;
  return ckeConfig;
})
```

That is all you need to know about configuring CKE in Neos, Refer to CKeditor5 documentation for more details on what you can do with it: https://docs.ckeditor.com/ckeditor5/latest/index.html

Richtext Toolbar

Way to retrieve:

```
globalRegistry.get('ckEditor5').get('richtextToolbar')
```

Contains the Rich Text Editing Toolbar components.

Buttons in the Rich Text Editing Toolbar are just plain React components.

The only way for these components to communicate with CKE is via its commands mechanism (@see https://docs.ckeditor.com/ckeditor5/latest/framework/guides/architecture/core-editor-architecture.html#commands)

Some commands may take arguments. Commands also contain state that is serialized into *formattingUnderCursor* redux state. Commands are provided and handled by CKE plugins, which may be registered via the configuration registry explained above.

The values are objects of the following form:

```
{
    commandName: 'bold' // A CKE command that gets dispatched
    commandArgs: [arg1, arg2] // Additional arguments passed together with a command
    component: Button // the React component being used for rendering
    isVisible: (editorOptions, formattingUnderCursor) => true // A function that␣
↪decides is the button should be visible or not
```

(continues on next page)

(continued from previous page)

```
    isActive: (formattingUnderCursor, editorOptions) => true // A function that␣
↪decides is the button should be active or not
    callbackPropName: 'onClick' // Name of the callback prop of the Component which is
                                fired when the component's value changes.

    // all other properties are directly passed on to the component.
}
```

4.14.6 CKEditor4-specific registries

Formatting rules

Way to retrieve:

```
globalRegistry.get('ckEditor').get('formattingRules')
```

Contains the possible styles for CKEditor.

Enabled Styles

The actual *enabled* styles are determined by the NodeTypes configuration of the property. This means, that if the node is configured in *NodeTypes.yaml* using:

```
properties:
  [propertyName]:
    ui:
      inline:
        editorOptions:
          formatting:
            strong: true
```

then the "strong" key inside this registry is actually enabled for the editor.

For backwards compatibility reasons, the formatting-and-styling-registry *KEYS* must match the "pre-React" UI, if they existed beforehand.

Configuration of CKEditor

With this config, CKEditor itself is controlled:

- the Advanced Content Filter (ACF) is configured, thus determining which markup is allowed in the editors
- which effect a button action actually has.

Currently, there exist three possible effects:

- triggering a command
- setting a style
- executing arbitrary code

Configuration Format

NOTE: one of "command" or "style" must be specified in all cases.

- *command* (string, optional). If specified, this CKEditor command is triggered; so the command string is known by CKEditor in the "commands" section: http://docs.ckeditor.com/#!/api/CKEDITOR.editor-method-getCommand
- *style* (object, optional). If specified, this CKEditor style is applied. Expects a style description adhering to CKEDITOR.style(...), so for example: *{ style: {element: 'h1'}*
- *config* (function, optional): This function needs to adjust the CKEditor config to e.g. configure ACF correctly. The function gets passed in the config so-far, AND the configuration from the node type underneath *ui.inline.editorOptions.formatting.[formatingRuleName]* and needs to return the modified config. See "CKEditor Configuration Helpers" below for helper functions.
- *extractCurrentFormatFn* (function, optional): If specified, this function will extract the current format. The function gets passed the currend "editor" and "CKEDITOR".
- *applyStyleFn* (function, optional): This function applies a style to CKEditor. Arguments: formattingOptions, editor, CKEDITOR.

CKEditor Configuration Helpers

- *config: registry.ckEditor.formattingRules.config.addToFormatTags('h1')*: adds the passed-in tag to the *format_tags* configuration option of CKEditor.
- *registry.ckEditor.formattingRules.config.add('Strong')*: adds the passed-in *Button Definition Name* to the ACF configuration (automatic mode). This means the button names are standard CKEditor config buttons, like "Cut,Copy,Paste,Undo,Redo,Anchor".

Richtext Toolbar

Contains the Rich Text Editing Toolbar components.

The values are objects of the following form:

```
{
  formattingRule: 'h1' // References a key inside "formattingRules"
  component: Button // the React component being used for rendering
  callbackPropName: 'onClick' // Name of the callback prop of the Component which is␣
↪fired when the component's value changes.

  // all other properties are directly passed on to the component.
}
```

Component wiring

- Each toolbar component receives all properties except "formattingRule" and "component" directly as props.
- Furthermore, the "isActive" property is bound, which is a boolean flag defining whether the text style referenced by "formatting" is currently active or not.
- Furthermore, the callback specified in "callbackPropName" is wired, which toggles the value.

For advanced use-cases; also the "formattingRule" is bound to the component; containing a formatting-rule identifier (string). If you need this, you'll most likely need to listen to *selectors.UI.ContentCanvas.formattingUnderCursor* and extract your relevant information manually.

Plugins

Way to retrieve:

```
globalRegistry.get('ckEditor').get('plugins')
```

Contains custom plugins for CkEditor.

```
plugins.set('plugin_key', {
    initFn: pluginInitFunction
});
```

pluginInitFunction is passed from CKEDITOR as the first argument. In that function you may register your plugin with CKEditor via its API (*CKEDITOR.plugins.add*). Take custom plugins as examples.

4.14.7 Data Loaders

Way to retrieve:

```
globalRegistry.get('dataLoaders')
```

A "Data Loader" controls asynchronous loading of secondary data, which is used in all kinds of Select / List boxes in the backend.

Example of data which is loaded through a data loader:

- Link Labels (in the inline link editor)
- Reference / References editor
- Data Sources in the Select Editor

Each Data Loader can have a slightly different API, so check the "description" field of each data loader when using it. It is up to the data loaders to implement caching internally.

Normally, each data loader exposes the following methods:

```
resolveValue(options, identifier) {
  // "options" is a DataLoader-specific object.
  // returns Promise with [{identifier, label}, ...] list; where "identifier" was␣
↪resolved to the actual object represented by "identifier".
}

search(options, searchTerm) {
  // "options" is a DataLoader-specific object.
```

(continues on next page)

(continued from previous page)

```
  // returns Promise with [{identifier, label}, ...] list; these are the objects␣
↪displayed in the selection dropdown.
}
```

4.14.8 Containers

Way to retrieve:

```
globalRegistry.get('containers')
```

The whole user interface is built around container components. They are registered through the containers registry. Below you will find an example on how to replace the PageTree container with your custom container:

```
manifest('Example', {}, globalRegistry => {
  const containerRegistry = globalRegistry.get('containers');
  containerRegistry.set('LeftSideBar/Top/PageTreeToolbar', () => null);
  containerRegistry.set('LeftSideBar/Top/PageTreeSearchbar', () => null);
  containerRegistry.set('LeftSideBar/Top/PageTree', FlatNavContainer);
});
```

4.14.9 Server Feedback Handlers

Way to retrieve:

```
globalRegistry.get('serverFeedbackHandlers')
```

Contains all server feedback handlers.

The key is the server-feedback-handler-type, and the value is a function with the following signature:

```
(feedback, store) => {
  // do whatever you like here :-)
}
```

4.14.10 Reducers

Way to retrieve:

```
globalRegistry.get('reducers')
```

Allows to register custom reducers for your plugin. It is probably a bad idea to override any of the existing reducers.

4.14.11 Sagas

Way to retrieve:

```
globalRegistry.get('sagas')
```

Allows to register custom sagas for your plugin. It is probably a bad idea to override any of the existing reducers.

Example:

```
function* watchNodeFocus() {
  yield takeLatest(actionTypes.CR.Nodes.FOCUS, function* (action) {
    yield put(actions.UI.FlashMessages.add(
      'testMessage',
      'Focused: ' + action.payload.contextPath,
      'success'
    ));
  });
}
manifest('The.Demo:Focus', {}, globalRegistry => {
  const sagasRegistry = globalRegistry.get('sagas');
  sagasRegistry.set('The.Demo/watchNodeFocus', {saga: watchNodeFocus});
});
```

4.15 Writing Tests For Neos

Testing and quality assurance documentation for Neos.

4.15.1 Behat tests for Neos

Setting up Neos for running Behat tests

The Neos package contains a growing suite of Behat tests which you should take into account while fixing bugs or adding new features. Please note that running these tests require that the Neos demo site package (Neos.Demo) is installed and activated.

Install Behat for the base distribution

Behat is installed in a separate folder and has a custom composer root file. To install Behat run the following composer command in *FLOW_ROOT/Build/Behat*:

```
cd Build/Behat
composer install
```

A special package *Neos.Behat* is used to integrate Flow with Behat and is installed if the base distribution was installed with *composer install –dev*.

Create configuration for subcontexts

Behat needs two special Flow contexts, *Development/Behat* and *Testing/Behat*.

- The context *Development/Behat* should be mounted as a separate virtual host and is used by Behat to do the actual HTTP requests.
- The context *Testing/Behat* is used inside the Behat feature context to set up test data and reset the database after each scenario.

These contexts should share the same database to work properly. Make sure to create a new database for the Behat tests since all the data will be removed after each scenario.

FLOW_ROOT/Configuration/Development/Behat/Settings.yaml:

```
Neos:
  Flow:
    persistence:
      backendOptions:
        dbname: 'neos_testing_behat'
```

FLOW_ROOT/Configuration/Testing/Behat/Settings.yaml:

```
Neos:
  Flow:
    persistence:
      backendOptions:
        dbname: 'neos_testing_behat'
        driver: pdo_mysql
        user: ''
        password: ''
```

Example virtual host configuration for Apache:

```
<VirtualHost *:80>
        DocumentRoot "FLOW_ROOT/Web"
        ServerName neos.behat.test
        SetEnv FLOW_CONTEXT Development/Behat
</VirtualHost>
```

Configure Behat

The Behat tests for Neos are shipped inside the Neos.Neos package in the folder *Tests/Behavior*. Behat uses a configuration file distributed with Neos, *behat.yml.dist*, or a local version, *behat.yml*. To run the tests, Behat needs a base URI pointing to the special virtual host running with the *Development/Behat* context. To set a custom base URI the default file should be copied and customized:

```
cd Packages/Application/Neos.Neos/Tests/Behavior
cp behat.yml.dist behat.yml
# Edit file behat.yml
```

Customized *behat.yml*:

```
default:
  paths:
    features: Features
```

(continues on next page)

(continued from previous page)

```
    bootstrap: %behat.paths.features%/Bootstrap
  extensions:
    Behat\MinkExtension\Extension:
      files_path: features/Resources
      show_cmd: 'open %s'
      goutte: ~
      selenium2: ~

      base_url: http://neos.behat.test/
```

Selenium

Some tests require a running Selenium server for testing browser advanced interaction and JavaScript. Selenium Server can be downloaded at http://docs.seleniumhq.org/download/ and started with:

```
java -jar selenium-server-standalone-2.x.0.jar
```

If using Saucelabs, you do not need your own Selenium setup.

Running Behat tests

Behat tests can be run from the Flow root folder with the *bin/behat* command by specifying the Behat configuration file:

```
bin/behat -c Packages/Application/Neos.Neos/Tests/Behavior/behat.yml
```

In case the executable file *bin/behat* is missing, create a symlink by running the following command in *FLOW_ROOT/bin*:

```
ln -s ../Build/Behat/vendor/behat/behat/bin/behat
```

Tip: You might want to warmup the cache before you start the test. Otherwise the tests might fail due to a timeout. You can do that with *FLOW_CONTEXT=Development/Behat ./flow flow:cache:warmup*.

Debugging

- Make sure to use a new database and configure the same databse for *Development/Behat* and *Testing/Behat*
- Run Behat with the *-v* option to get more information about errors and failed tests
- A failed step can be inspected by inserting "Then show last response" in the *.feature* definition

Run Behat tests on several browsers using Saucelabs

Note: Make sure that your Behat version is uptodate. Otherwise the saucelabs connection won't work. The *behat/mink-extension* need to be at least version 1.3.

Saucelabs (http://saucelabs.com) provides a VM infrastructure you can use to run your selenium tests on.

Using this infrastructure you can run the @javascript tagged tests on several Browsers and OSs autmatically without setting up your own selenium infrastructure.

To run Neos Behat tests with saucelabs you need to do the following steps.

Configure Behat

To talk to saucelabs you need to uncomment the following lines in the *behat.yml* and add your saucelabs username and access_key:

```
javascript_session: saucelabs
  saucelabs:
    username: <username>
    access_key: <access_key>
```

Tip: Saucelabs provides unlimited video time for Neos core development. If you want to contribute to Neos by writing tests ask Christian Müller.

To make tests with more browsers than the default browser you need to tell saucelabs which browser, version and OS you want to test on. You can add several browsers, each in its own profile. There are a lot of browsers configured already in the *saucelabsBrowsers.yml* file. You can include that into your behat configuration:

```
imports:
  - saucelabsBrowsers.yml
```

Open a tunnel to saucelabs

If you want to run the tests on your local machine you need to open a tunnel to saucelabs. This can be easily done by downloading Sauce Connect at https://docs.saucelabs.com/reference/sauce-connect/ and follow the instructions to setup and start it.

Run Behat tests

A test with Internet Explorer 10 on Windows8 would look like this then:

```
bin/behat -c Packages/Application/Neos.Neos/Tests/Behavior/behat.yml --profile␣
↪windows8-ie-10
```

You might just want to run the tests that need javascript on different browsers (all other tests won't use a browser anyways). Limit the tests to the @javascript tagged to do so:

```
bin/behat -c Packages/Application/Neos.Neos/Tests/Behavior/behat.yml --tags␣
↪javascript --profile windows8-ie-10
```

Note: The possible configuration settings for browsers can be found at https://saucelabs.com/docs/platforms. Choose "WebDriver" and "php" and click on the platform/browser combination you are interested in.

CHAPTER

FIVE

INSIDE OF NEOS

5.1 User Interface Development

These are the user interface development guidelines of Neos.

5.1.1 General User Interface Principles

The following principles serve as general guiding concepts throughout the whole Neos product.

Overall User Interface Goals

We have set up the following goals to strive for UI-wise:

- Reliable editing
- Predictable UI Behavior
- Immediate feedback for the user
- Built with the web - for the web

UI concepts should be evaluated against the above goals.

Technical guidelines / Goals

When implementing the user interface, we should follow these guidelines on a technical side:

- Take the pragmatic approach
- Augment the frontend website
- No iFrame in the content module, generally no iFrames except for bigger modal dialogs
- Browser support >= IE9; in the prototyping phase focus on Chrome / Firefox
- No polling of data from the server!
- A reload should always take you back to a safe state

CSS Guidelines

Overall Goal:

- Be pragmatic! We strive for solutions which work out-of-the-box in 95% of the cases; and tell the integrator how to solve the other 5%. Thus, the integrator has to care to make his CSS work with Neos; we do not use a sandbox.

Implementation notes:

- All CSS selectors should be fully lowercase, with - as separator. Example: `neos-menu`, `neos-inspector`
- We use the `neos-` prefix
- The integrator is never allowed to override `neos-`, `typo3-` and `aloha-`
- The main UI elements have an ID, and a partial reset is used to give us predictable behavior inside them.
- We use *sass*. To install, use +gem install sass compass+. Then, before modifying CSS, go to css/ and run +sass –compass –watch style.scss:style.css+. This will update style.css at every modification of style.scss.
- We use r.js for generating the Includes-built.css file. The command used by the built server is `r.js -o cssIn=Includes.css out=Includes-built.css`

Z-Indexes

The Neos UI uses Z-Indexes starting at *10000*.

Warning: TODO: Formulate some more about the usage of z-indexes.

5.1.2 Content Module Principles

In the Content Module, we directly render the *frontend* of the website, and then augment it with the Neos Content Editing User Interface.

Because of this, we do not control all CSS or javaScript which is included on the page; so we need some special guidelines to deal with that. These are listed on this page.

Naming of main UI parts

The following image shows the main UI parts of the content module and the names we use for them.

Content Module Architecture

The whole Content Module is built around the *Aloha Blocks*. Blocks are un-editable elements of a website, which are managed by Aloha. They can appear inside or outside editables, can be nested, and can appear either as inline element (`<span>`) or as block-level element(`<div>`).

Only one block is active at any given time. When a block is *active*, then all its parent blocks are *selected*. The *block selection* contains the active block as first element and all other selected blocks from innermost to outermost.

Most of the UI changes depending on the current block selection.

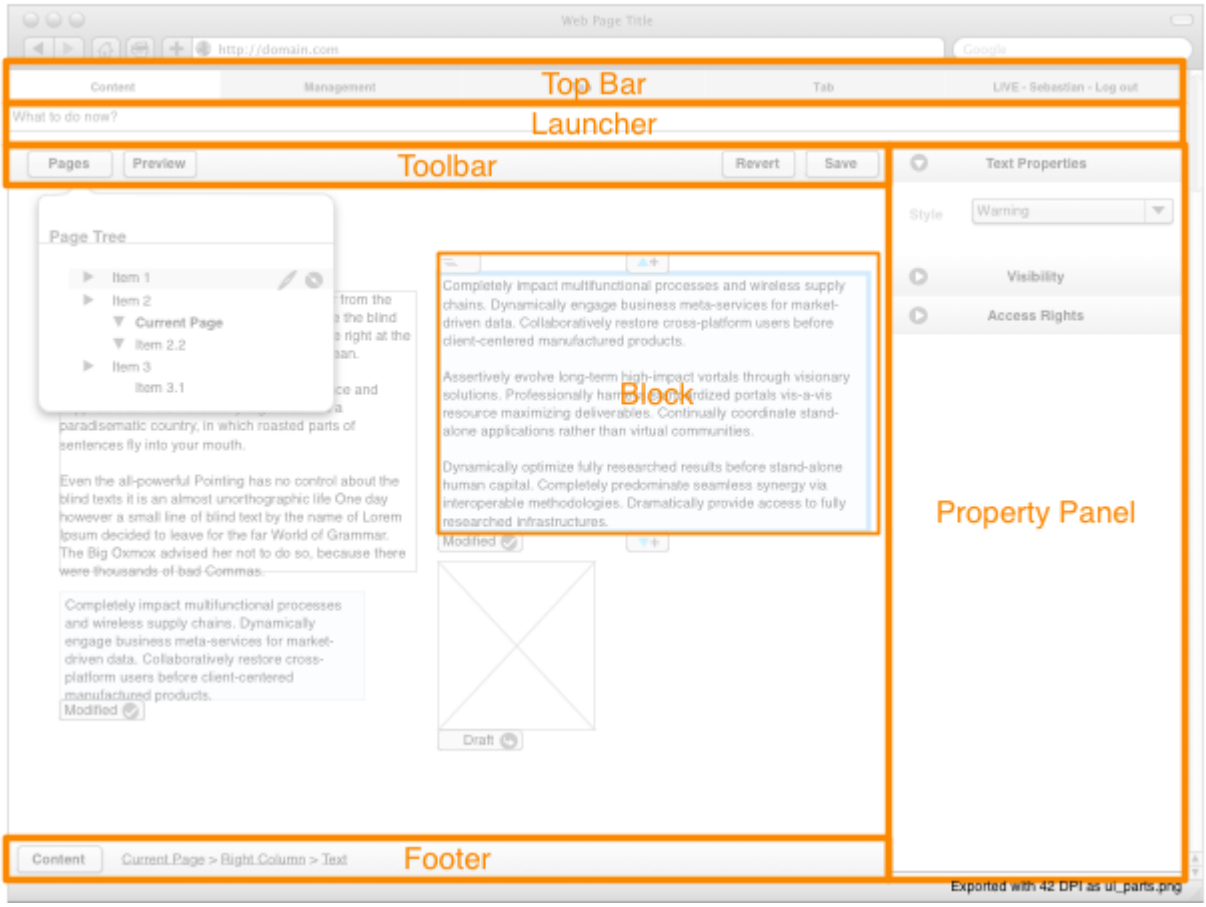


Fig. 1: UI parts of the content module

User Interface Updates on Selection Change

The following diagram shows how the UI is changing when the block selection changes:

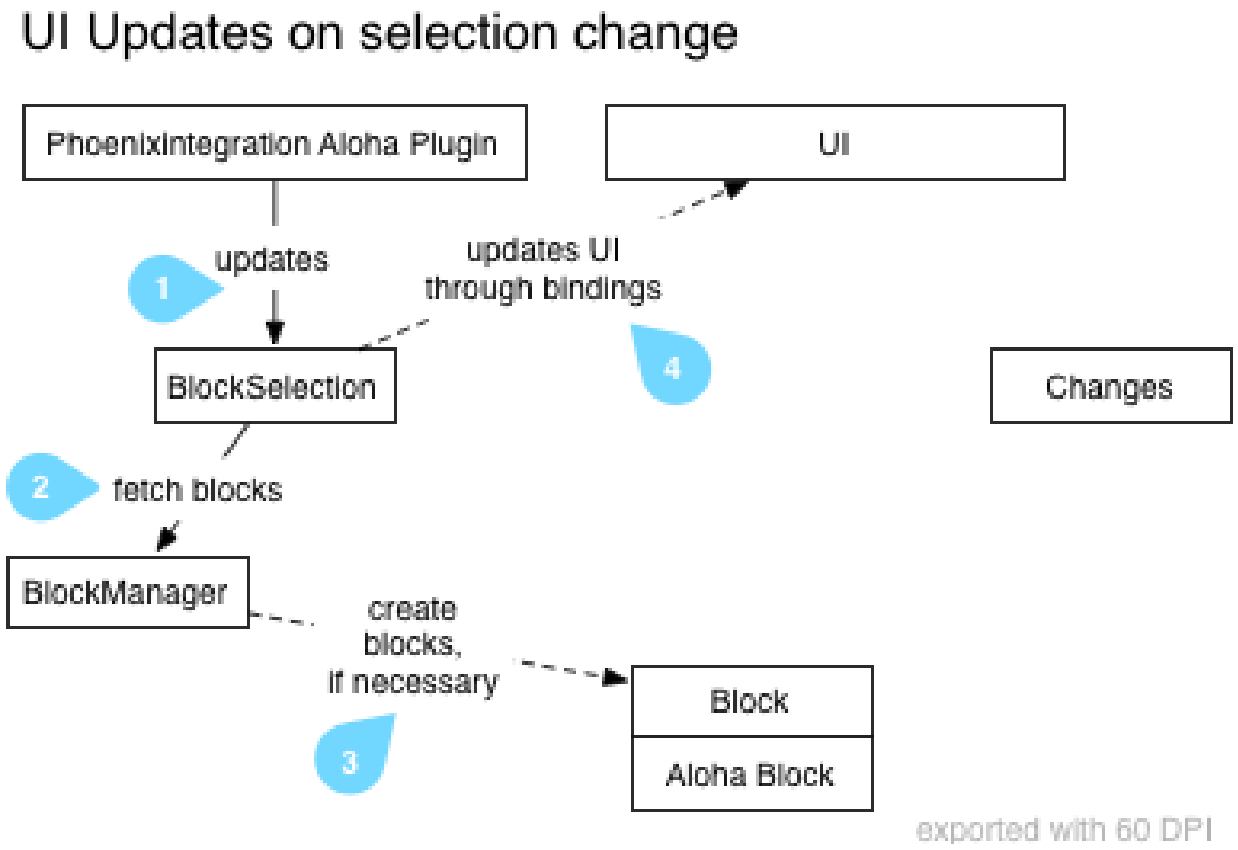


Fig. 2: UI Updates on selection change

1. The neosintegration Aloha Plugin (located in `alohaplugins/neosintegration/lib/neosintegration-plugin.js`) hooks into the Aloha event which is triggered whenever the block selection changes. Whenever this event is triggered, it calls `T3.Content.Model.BlockSelection.updateSelection()`.
2. We need to wrap each Aloha Block with a *Ember.js Block* (later only called Block), so we can attach event listeners to it. This wrapping is done by the `BlockManager`
3. The `BlockManager` either returns existing Ember.js Blocks (if the given Aloha Block has already been wrapped), or creates a new one.
4. Then, the `BlockSelection` sets its `content` property, which the UI is bound to. Thus, all UI elements which depend on the current block selection are refreshed.

User Interface Updates updates on property change

When an attribute is modified through the property panel, the following happens:

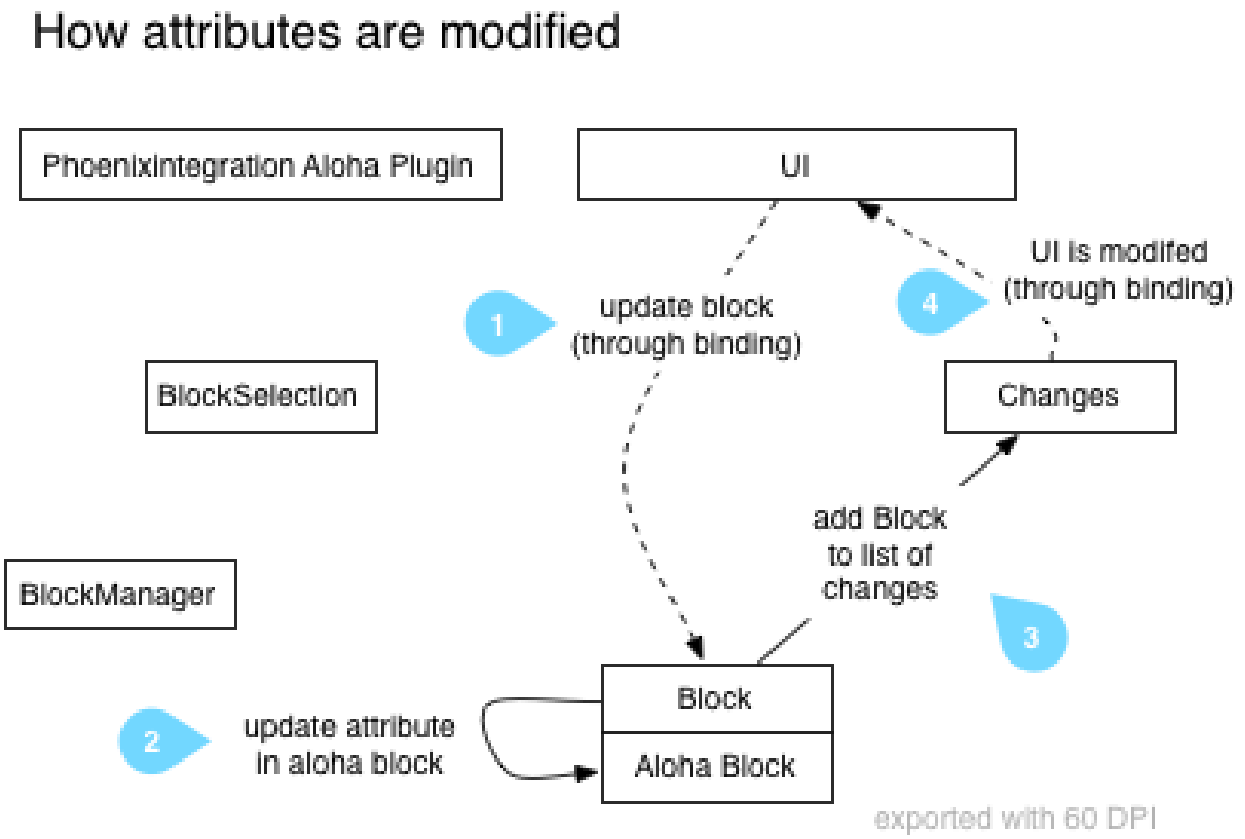


Fig. 3: How attributes are modified

WARNING: TODO: Document what happens when an *editable* is modified

Node Property Naming Conventions

TODO write some intro text

1. *Normal properties*

   Those properties contain normal values like a title, date or other value. Serverside setting of the property is done using Neos ContentRepository Node::setProperty()

2. *Visibility / lifecycle properties*

   These properties are prefixed using an underscore, like '_hidden'. Serverside setting of the property is done using Neos ContentRepository Node::set<UpperCamelCasePropertyname>()

3. *Neos internal properties*

   These properties are prefixed with a double underscore, like __workspacename TODO: internal

Saving content

Saving is triggered using `T3.Content.Model.Changes.save()` and is very straight-forward. For now, we use ExtDirect to send things back to the server.

Displaying Modal Dialogs

WARNING: TODO - write this

- REST architectural style
- HTML snippets loaded via fixed URLs from server side
- Return Commands (`<a rel="typo3-...." />`)

REST Server Side API

Most backend services which are currently used in the user interface are not RESTful. The goal is to migrate them, step by step, to a clean REST architecture.

Two services have been – partially – migrated: Nodes and ContentDimensions. We provide an HTML and a JSON based interface, roughly following HATEOAS concepts. Both formats a not yet part of the public API and we expect them to change as we gain more experience with the pros and cons of their structure.

URL `/neos/service/nodes` URL `/neos/service/contentdimensions`

Inspect the HTML output and the controller / template code for more information about the currently supported operations and arguments.

5.1.3 Backend Module Principles

For backend modules (that is, every module except the *content* area), we use the following guiding principles in addition to the already-existing principles:

- It should be possible to write backend modules only with PHP, without JavaScript involved
- Some features might be only available to the user if he has JavaScript enabled
- In order to introduce rich behavior, use the technique of *progressive enhancement*

Progressive Enhancement

As we want to use progressive enhancement heavily, we need to define some rules as a basis for that.

First, you should always think about the non-javascript functionality, and develop the feature without JavaScript enabled. This helps to get the client-server communication function correctly.

For most parts, you should not rely at all on any server state, but instead use URI parameters to encode required state. This makes the server-side code a lot easier and progressive enhancement more predictable.

Furthermore, if you reload certain parts of the user interface using AJAX, make sure to *always update the browser's URI* using History Management: In case there is an error, the user can just re-load the page and will get pretty much the same User Interface state. This fulfills our UI goal of "predictable UI behavior".

Connecting JavaScript code to the HTML content

In order to connect JavaScript code to HTML content, we (of course) rely on CSS selectors for finding the correct DOM nodes. However, we do *not* want to use CSS class attributes, as they change more frequently. Instead, we'd like to use special data-attributes to connect the JavaScript code to the user interface.

Note: In a nutshell:

- **CSS classes** are used for the **visible styling** only
- **HTML5 Data Attributes** are used for **connecting the JavaScript** code to HTML

We use the following data attributes for that:

- `data-area` is used to search for DOM nodes, for later usage in JavaScript.

  As an example, use `<div class="foo" data-area="actionBar"></div>` in the HTML and match it using `$('[data-area=actionBar]')` in JavaScript.

- `data-json` is used for transferring server-side state to the JavaScript as JSON.

  Example: We need the full URI parameters which have been used for the current rendering as array/object on the client side. Thus, the server side stores them inside `<div style="display:none" data-json="uriParameters">{foo: 'bar'}</div>`.

  The JavaScript code then accesses them at a central place using `JSON.parse($('[data-json=uriParameters]').text())` and makes them available using some public API.

- `data-type` is used to mark that certain parts of the website contain a client-side template language like handlebars.

  As an example for the action bar, we use the following code here:

```
  <button>
     Edit
     <span class="js" data-type="handlebars">
        {{#if multipleSelectionActive}} {{numberOfSelectedElements}} elements{{/
↪if}}
     </span>
  </button>

Then, on the client side in JavaScript, we use the handlebars template␣
↪accordingly.
```

Adjusting the UI if JavaScript is (in-)active

Often, you want to hide or show some controls depending on whether JavaScript is enabled or disabled. By default, every DOM element is visible no matter whether JavaScript is enabled or not.

If you want to show a DOM element only if JavaScript is **enabled**, use the CSS class `js`.

If you want to show a DOM element only if JavaScript is **disabled**, use the CSS class `nojs`.

5.1.4 JavaScript Style Guide

Code Conventions

- We use only the `TAB` character for indentation.
- We use `UpperCamelCase` for class names, and `lowerCamelCase` for method and property names.
- Methods and properties that begin with an underscore (_) are `private`.
- Variables which contain *jQuery* elements should be named like `$element`, starting with a `$`. If it is a private jQuery element, prefix it with `_$`
- We use `that` as a name for a closure reference to `this`, but try to avoid it if there's the possibility of scope binding. Unfortunately jQuery's event handlers do not allow easy scope binding.

Code Documentation

TODO: still determine this.

RequireJS module skeleton

All JavaScript files are RequireJS modules. They should follow this structure:

WARNING: still has to be done and discussed

<javascript> TODO </javascript>

Public API, Private methods and attributes

All methods and properties which are public API are marked with `@api`. The public API is supported for a longer period, and when a public API changes, this is clearly communicated in the Release Notes of a version.

On the contrary, we prefix `private` methods and attributes with an underscore. The *user* of an API should never override or call methods `private` methods as they are not meant for him to be overridden.

There's also a type in between: methods which are not `private` but do not have a `@api` annotation. They can be safely overridden by the user, and he should not experience any unwanted behavior. Still, the names or functionality of these methods can change without notice between releases. In the long run, all of these methods should become part of the public API, as soon as they are proven in real life.

To sum it up, we have three types of methods/properties:

- `@api` methods: Public API, the user of the object can rely on the functionality to be stable, changes in @api are clearly communicated
- non-`@api` but also not private: The user can use it, but needs to be aware the method might still change.
- private (prefixed with _): The user should never ever call or access this. Very strange things might happen.

Note: It is allowed to observe or bind to private properties within the Neos javascript code. This is because the property is not just meant as private object property, but as a non-api property.

When to use a new file

JavaScript files can become pretty large, so there should be a point to create a new file. Having just one class per file would be too much though, as this would end up in possibly hundreds of files, from which a lot will just have 20 lines of code.

As we use requirejs for loading dependencies we came up with the following guidelines for creating a new file:

- Classes using a template include using the `!text` plugin should be in a separate file
- If a class is extended by another class, then it should be in a separate file so it can be easily loaded as dependency
- If a class is huge, and affecting readability of the definition file, then it should be moved to a single file
- It has preference to keep classes grouped together, so classes with just a few lines stay grouped together, so if none of the above is true the classes stays in the main file.

5.1.5 Ember.JS Tips & Tricks

Dealing with classes and objects

- Always extend from `Ember.Object` (or a subclass)
- Extension is done using `Ember.Object.extend({...})`
- Never use `new` to instantiate new objects. Instead, use `TheObject.create(...)`
- All objects have generic `set(key, value)` and `get(key)` methods, *which should be used under all circumstances*!

The following example shows this:

```
var Foo = Ember.Object.extend({
        someValue: 'hello',
        myMethod: function() {
                alert(this.get('someValue'));
        }
});

var fooInstance = Foo.create({
        someValue: 'world'
});
fooInstance.myMethod(); // outputs "world"
```

Inheritance can be used just as in PHP, since Emberjs binds a special `._super()` function for every method call (in fact the function is wrapped to create this special `_super` method). So calling the current method of the superclass can be done without specifying the superclass and method name.

```
var Foo = Ember.Object.extend({
        greet: function(name) {
                return 'Hello, ' + name;
        }
});
var Bar = Foo.extend({
        greet: function(name) {
                return 'Aloha and ' + this._super(name);
        }
});
```

(continues on next page)

(continued from previous page)

```
Bar.create().greet('Neos'); // outputs "Aloha and Hello, Neos"
```

Data Binding tips and tricks

To create a *computed property*, implement it as function and append +.property()+:

```
var Foo = Ember.Object.extend({
        someComputedValue: function() {
                return "myMethod";
        }.property()
});
```

If your computed property reads other values, specify the dependent values as parameters to `property()`. If the computed property is deterministic and depends only on the dependant values, it should be marked further with `.cacheable()`.

```
var Foo = Ember.Object.extend({
        name: 'world',
        greeting: function() {
                return "Hello " + this.attr('name');
        }.property('name').cacheable()

});
```

Now, every time `name` changes, the system re-evaluates `greeting`.

Note: Forgetting `.cacheable()` can have severe performance penalties and result in circular loops, in worst case freezing the browser completely.

You can also use a getter / setter on a property, if you do this it's **extremely important to return the value of the property** in the setter method.

```
var Foo = Ember.Object.extend({
        firstName: null,
        lastName: null,

        fullName: function(key, value) {
                if (arguments.length === 1) {
                        return this.get('firstName') + ' ' + this.get('lastName');
                } else {
                        var parts = value.split(' ');
                        this.set('firstName', parts[0]);
                        this.set('lastName', parts[1]);

                        return value;
                }
        }.property('firstName', 'lastName').cacheable()
});
```

Observe changes

To react on changes of properties in models or views (or any other class extending `Ember.Observable`), a method marked as an observer can be used. Call `.observes('propertyName')` on a private method to be notified whenever a property changes.

```
var Foo = Ember.Object.extend({
        name: 'world',
        _nameDidChange: function() {
                console.log('name changed to', this.get('name'));
        }.observes('name')
});
```

5.1.6 Translating the user interface

Default Language

The `availableLanguages` are defined in `Packages/Application/Neos.Neos/Configuration/Settings.yaml`.

You may override the default language of your installation in `Configuration/Settings.yaml`:

```
Neos:
  Neos:
    userInterface:
      defaultLanguage: 'en'
```

Label Scrambling

To help you find labels in the Neos editor interface that you still need to translate, you can use the language label scrambling setting in your yaml file. This will replace all translations by a string consisting of only # characters with the same length as the actual translated label. With this setting enabled every still readable string in the backend is either content or non-translated.

```
Neos:
  Neos:
    userInterface:
      scrambleTranslatedLabels: TRUE
```

Note: The translation labels used in the javascript ui are parsed to a big json file. While changing xliff files this cached should be flushed, but still it can turn out useful to disable this cache. You can do so by using the following snippet in your *Caches.yaml*

```
Neos_Neos_XliffToJsonTranslations:
  backend: Neos\Flow\Cache\Backend\NullBackend
```

CHAPTER

SIX

REFERENCES

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

6.1 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 Client-side Configuration Processing* for more details.

6.1.1 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.

6.1.2 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.'
```

(continues on next page)

(continued from previous page)

```
    inspector:
      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.

6.1.3 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**

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

In case a lot of space is needed for the text (f.e. for HTML source code), 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.

6.1.5 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:
      group: 'document'
      editor: 'Neos.Neos/Inspector/Editors/SelectBoxEditor'
      editorOptions:
        values:
          firstChildNode:
```

(continues on next page)

(continued from previous page)

```
            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'

        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: 'Content/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"
},
{
  "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.

6.1.6 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` node 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
```

(continues on next page)

(continued from previous page)

```
        nodeTypes: ['Neos.Neos:Shortcut']
        placeholder: 'Paste a link, or type to search for nodes'
```

Options Reference:

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

6.1.7 Property Type: integer `TextFieldEditor`

Example:

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

Options Reference:

all TextFieldEditor options apply

6.1.8 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.

6.1.9 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.

6.1.10 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'
```

(continues on next page)

(continued from previous page)

```
      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:
        accept: 'image/png'
        features:
          crop: true
        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:
        accept: '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) Set the accepted mime type for this editor. If non is given it falls back to `image/*`.

`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 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.

6.1.11 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` Set the accepted mime type for this editor. If non is given all files are allowed.

`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 and 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*.

6.2 View Helper Reference

6.2.1 Content Repository ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

PaginateViewHelper

This ViewHelper renders a Pagination of nodes.

Implementation Neos\ContentRepository\ViewHelpers\Widget\PaginateViewHelper

Arguments

- `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*): Additional configuration
- `widgetId` (string, *optional*): Unique identifier of the widget instance

6.2.2 FluidAdaptor ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

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

- `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

Examples

Defaults:

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

Expected result:

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

Defaults:

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

Expected result:

```
1,023.00 B
// 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 uppercase 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*)

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>
    </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

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

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.
- `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)
- `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

f:widget.paginate

This ViewHelper renders a Pagination of objects.

Implementation Neos\FluidAdaptor\ViewHelpers\Widget\PaginateViewHelper

Arguments

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

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, *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)

6.2.3 Form ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

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.
- `dateFormat` (string, *optional*)
- `enableDatePicker` (boolean, *optional*)
- `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

neos.form:form.formElementRootlinePath

Form Element Rootline Path

Implementation Neos\Form\ViewHelpers\Form\FormElementRootlinePathViewHelper

Arguments

- `renderable` (NeosFormCoreModelRenderableRenderableInterface)

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.
- `as` (string, *optional*)
- `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.

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.
- `as` (string, *optional*)
- `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.

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*): name of the preset to use

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*): If specified, only the values of the given renderable are rendered, otherwise all form elements are rendered
- `formRuntime` (NeosFormCoreRuntimeFormRuntime, *optional*): If not set, the Form Runtime will be fetched from the View, which only works within the FluidFormRenderer
- `as` (string, *optional*)

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)
- `element` (NeosFormCoreModelFormElementInterface, *optional*)

6.2.4 Fusion ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

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*): Additional 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)
```

6.2.5 Media ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

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.
- `file` (mixed, *optional*): The Asset object. DEPRECATED, use $asset instead!
- `asset` (mixed, *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` (mixed, *optional*)
- `height` (mixed, *optional*)
- `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

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.
- `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)
- `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

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` (DateTime, *optional*)

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.
- `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 image
- `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
- `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

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" />
```

(continues on next page)

(continued from previous page)

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.
- `asset` (NeosMediaDomainModelAssetInterface, *optional*): 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
- `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

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 retrieve the path from
- `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*): 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 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, *optional*)
- `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
```

6.2.6 Neos ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

neos:backend.authenticationProviderLabel

Renders a label for the given authentication provider identifier

Implementation Neos\Neos\ViewHelpers\Backend\AuthenticationProviderLabelViewHelper

Arguments

- `identifier` (string)

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*)
- `minimalBrightness` (integer, *optional*)

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)

neos:backend.container

ViewHelper for the backend 'container'. Renders the required HTML to integrate the Neos backend into a website.

Implementation Neos\Neos\ViewHelpers\Backend\ContainerViewHelper

Arguments

- `node` (NeosContentRepositoryDomainModelNodeInterface)

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): A node

neos:backend.interfaceLanguage

ViewHelper for rendering the current backend users interface language.

Implementation Neos\Neos\ViewHelpers\Backend\InterfaceLanguageViewHelper

neos:backend.javascriptBuiltVersion

Returns a shortened md5 of the built JavaScript file

Implementation Neos\Neos\ViewHelpers\Backend\JavascriptBuiltVersionViewHelper

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.shouldLoadMinifiedJavascript

Returns true if the minified Neos JavaScript sources should be loaded, false otherwise.

Implementation Neos\Neos\ViewHelpers\Backend\ShouldLoadMinifiedJavascriptViewHelper

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.
- `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.
- `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: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*): The node of the content element. Optional, will be resolved from the Fusion context by default.

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 determine 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.
- `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
- `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

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.
- `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 variable the node will be assigned to for the rendered child content
- `nodeVariableName` (string, *optional*): The name of the base node inside the Fusion context to use for the ContentContext or resolving relative paths

- `resolveShortcuts` (boolean, *optional*): INTERNAL Parameter - if false, shortcuts are not redirected to their target. Only needed on rare backend occasions when we want to link to the shortcut itself.
- `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

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)

neos:rendering.inBackend

ViewHelper to find out if Neos is rendering the backend.

Implementation Neos\Neos\ViewHelpers\Rendering\InBackendViewHelper

Arguments

- `node` (NeosContentRepositoryDomainModelNodeInterface, *optional*)

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>
```

(continues on next page)

(continued from previous page)

```
  <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*)

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` (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:

```
<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*)
- `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*): INTERNAL Parameter - if false, shortcuts are not redirected to their target. Only needed on rare backend occasions when we want to link to the shortcut itself.

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.)
```

6.2.7 TYPO3 Fluid ViewHelper Reference

This reference was automatically generated from code on 2019-03-05

6.3 Fusion Reference

6.3.1 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.

myArray = Neos.Fusion:Join {
        o1 = Neos.NodeTypes:Text
        o1.@position = 'start 12'
        o2 = Neos.NodeTypes:Text
```

(continues on next page)

(continued from previous page)

```
        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):

```
myArray = 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

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

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)

Neos.Fusion:Reduce

Reduce the given items to a single value by using `itemRenderer`.

items (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

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:Component

Create a component that adds all properties to the props context and afterward evaluates the renderer.

renderer (mixed, **required**) The value which gets rendered

Example:

```
prototype(Vendor.Site:Component) < prototype(Neos.Fusion:Component) {
        title = 'Hello World'
        titleTagName = 'h1'
        description = 'Description of the Neos World'
        bold = false

        renderer = Neos.Fusion:Tag {
                attributes.class = Neos.Fusion:DataStructure {
                        component = 'component'
                        bold = ${props.bold ? 'component--bold' : false}
                }
                content = Neos.Fusion:Join {
                        headline = Neos.Fusion:Tag {
                                tagName = ${props.titleTagName}
                                content = ${props.title}
                        }

                        description = Neos.Fusion:Tag {
                                        content = ${props.description}
                        }
                }
        }
}
```

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')}
                }
```

(continues on next page)

(continued from previous page)

```
        }

        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: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 (*Neos.Fusion:Attributes*) Tag attributes

Example:

```
htmlTag = Neos.Fusion:Tag {
        tagName = 'html'
        omitClosingTag = TRUE

        attributes {
                version = 'HTML+RDFa 1.1'
                xmlns = 'http://www.w3.org/1999/xhtml'
        }
}
```

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

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: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

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: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

Example:

```
canRender = Neos.Fusion:CanRender {
        type = 'My.Package:Prototype'
}
```

6.3.2 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">
```

(continues on next page)

(continued from previous page)

```
                        <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
```

(continues on next page)

(continued from previous page)

```
        # title = ${q(node).property('title')}
}
```

Neos.Neos:ContentComponent

Base type to render component based content-nodes, extends *Neos.Fusion:Component*.

renderer (mixed, **required**) The value which gets rendered

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'
}
```

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)

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) Men^u level the item is rendered on

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)}
}
```

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

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`

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'
        }
}
```

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'
        }
}
```

6.4 Eel Helpers Reference

This reference was automatically generated from code on 2019-03-05

6.4.1 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` (array|mixed) First array or value
- `array2` (array|mixed) Second array or value
- `array_` (array|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` (array) 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` (array) 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` (array) 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` (array)

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` (array) 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` (array) The array

Return (bool) true if the array is empty

Array.join(array, separator)

Join values of an array with a separator

- `array` (array) 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` (array) The array

Return (array)

Array.last(array)

Get the last element of an array

- `array` (array) The array

Return (mixed)

Array.length(array)

Get the length of an array

- `array` (array) 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` (array) 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` (array)

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` (array)
- `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` (array) 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` (array) The array

Return (array)

Array.set(array, key, value)

Set the specified key in the the array

- `array` (array)
- `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` (array)

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` (array)
- `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` (array) 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` (array) 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` (array)

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` (array)
- `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.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` (array)
- `element` (mixed)

Return (array) The array with the inserted elements

6.4.2 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)

6.4.3 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

6.4.4 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

6.4.5 File

Helper to read files.

Implemented in: `Neos\Eel\Helper\FileHelper`

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)

6.4.6 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)

6.4.7 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` (float) 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 (integer) The integral part of the given number

6.4.8 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)

6.4.9 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)

6.4.10 Neos.Link

Eel helper for the linking service

Implemented in: `Neos\Neos\Fusion\Helper\LinkHelper`

Neos.Link.convertUriToObject(uri, contextNode)

- `uri` (string|Uri)
- `contextNode` (NodeInterface, *optional*)

Return (NodeInterface|AssetInterface|NULL)

Neos.Link.getScheme(uri)

- `uri` (string|Uri)

Return (string)

Neos.Link.hasSupportedScheme(uri)

- `uri` (string|Uri)

Return (boolean)

Neos.Link.resolveAssetUri(uri)

- `uri` (string|Uri)

Return (string)

Neos.Link.resolveNodeUri(uri, contextNode, controllerContext)

- `uri` (string|Uri)
- `contextNode` (NodeInterface)
- `controllerContext` (ControllerContext)

Return (string)

6.4.11 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.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)

6.4.12 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)

6.4.13 Neos.Ui.Modules

Implemented in: `Neos\Neos\Ui\Fusion\Helper\ModulesHelper`

Neos.Ui.Modules.isAllowed(modulePath)

Checks whether the current user has access to a module

- `modulePath` (string)

Return (boolean)

Neos.Ui.Modules.isAvailable(moduleName)

Checks, whether a module is available to the current user

- `moduleName` (string)

Return (boolean)

Neos.Ui.Modules.isEnabled(modulePath)

Checks whether a module is enabled

- `modulePath` (string)

Return (boolean)

6.4.14 Neos.Ui.PositionalArraySorter

Implemented in: `Neos\Neos\Ui\Fusion\Helper\PositionalArraySorterHelper`

Neos.Ui.PositionalArraySorter.sort(array, positionPath)

- `array` (array)
- `positionPath` (string, *optional*)

Return (array)

6.4.15 Neos.Ui.Sites

Implemented in: `Neos\Neos\Ui\Fusion\Helper\SitesHelper`

Neos.Ui.Sites.isActive(siteNode)

6.4.16 Neos.Ui.StaticResources

Implemented in: `Neos\Neos\Ui\Fusion\Helper\StaticResourcesHelper`

Neos.Ui.StaticResources.compiledResourcePackage()

6.4.17 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)

6.4.18 NodeInfo

Implemented in: `Neos\Neos\Ui\Fusion\Helper\NodeInfoHelper`

NodeInfo.createRedirectToNode(controllerContext, node)

- `controllerContext` (ControllerContext)
- `node` (NodeInterface, *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)

6.4.19 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

6.4.20 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` (integer, *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)

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("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

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"
```

Note: this method does not perform regular expression matching, @see pregReplace().

- `string` (string) The input string
- `search` (string) A search string
- `replace` (string) A replacement string

Return (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, *optional*) The separator where the string should be splitted
- `limit` (integer, *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, *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

6.4.21 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) Translated label or source label / ID key

Translation.value(value)

Start collection of parameters for translation by original label

- `value` (string)

Return (TranslationParameterToken)

6.4.22 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 (stringlNULL)

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)

6.5 FlowQuery Operation Reference

This reference was automatically generated from code on 2019-03-05

6.5.1 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

6.5.2 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

6.5.3 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

6.5.4 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

6.5.5 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

6.5.6 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

6.5.7 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

6.5.8 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 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 value's last 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 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

6.5.9 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 Neos.NodeTypes: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

6.5.10 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 Neos.NodeTypes:Text]')

Example (multiple node types):

q(node).find('[instanceof Neos.NodeTypes:Text],[instanceof Neos.NodeTypes:Image]')

Example (node type with filter):

q(node).find('[instanceof Neos.NodeTypes:Text][text*="Neos"]')

Implementation Neos\ContentRepository\Eel\FlowQueryOperations\FindOperation

Priority 100

Final No

Returns void

6.5.11 first

Get the first element inside the context.

Implementation Neos\Eel\FlowQuery\Operations\FirstOperation

Priority 1

Final No

Returns void

6.5.12 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

6.5.13 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

6.5.14 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

6.5.15 last

Get the last element inside the context.

Implementation Neos\Eel\FlowQuery\Operations\LastOperation

Priority 1

Final No

Returns void

6.5.16 neosUiDefaultNodes

Fetches all nodes needed for the given state of the UI

Implementation Neos\Neos\Ui\FlowQueryOperations\NeosUiDefaultNodesOperation

Priority 100

Final No

Returns void

6.5.17 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

6.5.18 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

6.5.19 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

6.5.20 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

6.5.21 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

6.5.22 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

6.5.23 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

6.5.24 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\Neos\Eel\FlowQueryOperations\ParentsUntilOperation

Priority 100

Final No

Returns void

6.5.25 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\ContentRepository\Eel\FlowQueryOperations\ParentsUntilOperation

Priority 0

Final No

Returns void

6.5.26 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

6.5.27 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

6.5.28 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

6.5.29 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

6.5.30 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

6.5.31 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

6.5.32 search

Implementation Neos\Neos\Ui\FlowQueryOperations\SearchOperation

Priority 100

Final No

Returns void

6.5.33 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

6.5.34 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

6.5.35 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).

Implementation Neos\Neos\Eel\FlowQueryOperations\SortOperation

Priority 1

Final No

Returns mixed

6.6 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 2017-05-11

6.6.1 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

Will remove all undefined properties according to the node type configuration.

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).

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.

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 Neos.NodeTypes: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)

6.6.2 Package *NEOS.FLOW*

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 also removes any session data.

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: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.

./flow configuration:show –type Settings –path Neos.Flow.persistence

Options

--type Configuration type to show

--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 web-server 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:core:shell

Run the interactive Shell

The shell command runs Flow's interactive shell. This shell allows for entering commands like through the regular command line interface but additionally supports autocompletion and a user-based command history.

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

The main purpose of this is to fix setups that were created with Flow 2.3.x or earlier and whose database server did not have a default collation of utf8mb4_unicode_ci. 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. One thing this might be useful for is when switching to the utf8mb4mb4 character set, see:

- https://mathiasbynens.be/notes/mysql-utf8mb4
- https://florian.ec/articles/mysql-doctrine-utf8/

Note: This command **is not a general purpose conversion tool**. It will specifically not fix cases of actual utf8mb4 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

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

--show-descriptions Show descriptions for the migrations (enables versions display)

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:package:activate

Activate an available package

This command activates an existing, but currently inactive package.

Arguments

--package-key The package key of the package to create

Related commands

neos.flow:package:deactivate Deactivate a package

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:deactivate

Deactivate a package

This command deactivates a currently active package.

Arguments

--package-key The package key of the package to create

Related commands

neos.flow:package:activate Activate an available package

neos.flow:package:delete

Delete an existing package

This command deletes an existing package identified by the package key.

Arguments

--package-key The package key of the package to create

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 and its state – active or inactive.

Options

--loading-order The returned packages are ordered by their loading order.

Related commands

neos.flow:package:activate Activate an available package

neos.flow:package:deactivate Deactivate a package

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:getpath

Generate a route path

This command takes package, controller and action and displays the generated route path and the selected route:

./flow routing:getPath –format json Acme.Demo\Sub\Package

Arguments

--package Package key and subpackage, subpackage parts are separated with backslashes

Options

--controller Controller name, default is 'Standard'

--action Action name, default is 'index'

--format Requested Format name default is 'html'

neos.flow:routing:list

List the known routes

This command displays a list of all currently registered routes.

neos.flow:routing:routepath

Route the given route path

This command takes a given path and displays the detected route and the selected package, controller and action.

Arguments

--path The route path to resolve

Options

--method The request method (GET, POST, PUT, DELETE, ...) to simulate

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: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: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: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

6.6.3 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="http://typo3.org/ns/TYPO3/Fluid/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 "http://typo3.org/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.

6.6.4 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. This option can only be used in combination with –generate-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-related Also create the mentioned package, related model and repository if neccessary.

--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

Related commands

typo3.flow:package:create *Command not available*

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

6.6.5 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

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

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

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

6.6.6 Package *NEOS.NEOS*

neos.neos:domain:activate

Activate a domain record by hostname

Arguments

--hostname The hostname to activate

neos.neos:domain:add

Add a domain record

Arguments

--site-node-name The nodeName of the site rootNode, e.g. “neostypo3org

--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

Arguments

--hostname The hostname to deactivate

neos.neos:domain:delete

Delete a domain record by hostname

Arguments

--hostname The hostname to remove

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

This command activates the specified site.

Arguments

--site-node The node name of the site to activate

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.

If no `nodeType` option is specified the command will use *Neos.NodeTypes:Page* as fallback. The node type 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

Options

--node-type The node type to use for the site node. (Default = Neos.NodeTypes:Page)

--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

This command deactivates the specified site.

Arguments

--site-node The node name of the site to deactivate

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 all content and related data - for now. In the future we need some more sophisticated cleanup.

Options

--site-node Name of a site root node to clear only content of this site.

neos.neos:user:activate

Activate a user

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.

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

--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

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.

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

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

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

--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:discardall

Discard changes in workspace (DEPRECATED)

This command discards all modified, created or deleted nodes in the specified workspace.

Arguments

--workspace-name Name of the workspace, for example “user-john

Options

--verbose If enabled, information about individual nodes will be displayed

Related commands

neos.neos:workspace:discard Discard changes in workspace

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:publishall

Publish changes of a workspace (DEPRECATED)

This command publishes all modified, created or deleted nodes in the specified workspace to the live workspace.

Arguments

--workspace-name Name of the workspace, for example “user-john

Options

--verbose If enabled, information about individual nodes will be displayed

Related commands

neos.neos:workspace:publish Publish changes of a workspace

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

6.6.7 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

6.7 Validator Reference

6.7.1 Flow Validator Reference

This reference was automatically generated from code on 2019-03-05

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

---

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

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

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

---

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

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

---

6.7.2 Media Validator Reference

This reference was automatically generated from code on 2019-03-05

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)

6.7.3 Party Validator Reference

This reference was automatically generated from code on 2019-03-05

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

6.8 Signal Reference

6.8.1 Content Repository Signals Reference

This reference was automatically generated from code on 2019-03-05

Context (`Neos\ContentRepository\Domain\Service\Context`)

This class contains the following signals.

beforeAdoptNode

Autogenerated Proxy Method

afterAdoptNode

Autogenerated Proxy Method

Node (`Neos\ContentRepository\Domain\Model\Node`)

This class contains the following signals.

beforeNodeMove

Autogenerated Proxy Method

afterNodeMove

Autogenerated Proxy Method

beforeNodeCopy

Autogenerated Proxy Method

afterNodeCopy

Autogenerated Proxy Method

beforeNodeCreate

Autogenerated Proxy Method

afterNodeCreate

Autogenerated Proxy Method

nodeAdded

Autogenerated Proxy Method

nodeUpdated

Autogenerated Proxy Method

nodeRemoved

Autogenerated Proxy Method

beforeNodePropertyChange

Autogenerated Proxy Method

nodePropertyChanged

Autogenerated Proxy Method

nodePathChanged

Autogenerated Proxy Method

NodeData (`Neos\ContentRepository\Domain\Model\NodeData`)

This class contains the following signals.

nodePathChanged

Autogenerated Proxy Method

NodeDataRepository (`Neos\ContentRepository\Domain\Repository\NodeDataRepository`)

This class contains the following signals.

repositoryObjectsPersisted

Autogenerated Proxy Method

PublishingService (`Neos\ContentRepository\Domain\Service\PublishingService`)

This class contains the following signals.

nodePublished

Autogenerated Proxy Method

nodeDiscarded

Autogenerated Proxy Method

Workspace (`Neos\ContentRepository\Domain\Model\Workspace`)

This class contains the following signals.

baseWorkspaceChanged

Autogenerated Proxy Method

beforeNodePublishing

Autogenerated Proxy Method

afterNodePublishing

Autogenerated Proxy Method

6.8.2 Flow Signals Reference

This reference was automatically generated from code on 2019-03-05

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.

AbstractBackend (`Neos\Flow\Persistence\Generic\Backend\AbstractBackend`)

This class contains the following signals.

removedObject

Autogenerated Proxy Method

persistedObject

Autogenerated Proxy Method

ActionRequest (`Neos\Flow\Mvc\ActionRequest`)

This class contains the following signals.

requestDispatched

Autogenerated Proxy Method

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

loggedOut

Autogenerated Proxy Method

successfullyAuthenticated

Autogenerated Proxy Method

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

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

afterControllerInvocation

Autogenerated Proxy Method

DoctrineCommandController (`Neos\Flow\Command\DoctrineCommandController`)

This class contains the following signals.

afterDatabaseMigration

Autogenerated Proxy Method

EntityManagerFactory (`Neos\Flow\Persistence\Doctrine\EntityManagerFactory`)

This class contains the following signals.

beforeDoctrineEntityManagerCreation

Autogenerated Proxy Method

afterDoctrineEntityManagerCreation

Autogenerated Proxy Method

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

PersistenceManager (`Neos\Flow\Persistence\Generic\PersistenceManager`)

This class contains the following signals.

allObjectsPersisted

Autogenerated Proxy Method

PolicyService (`Neos\Flow\Security\Policy\PolicyService`)

This class contains the following signals.

configurationLoaded

Autogenerated Proxy Method

rolesInitialized

Autogenerated Proxy Method

SlaveRequestHandler (`Neos\Flow\Cli\SlaveRequestHandler`)

This class contains the following signals.

dispatchedCommandLineSlaveRequest

Emits a signal that a CLI slave request was dispatched.

6.8.3 Media Signals Reference

This reference was automatically generated from code on 2019-03-05

Asset (`Neos\Media\Domain\Model\Asset`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

AssetService (`Neos\Media\Domain\Service\AssetService`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

assetRemoved

Autogenerated Proxy Method

assetUpdated

Autogenerated Proxy Method

assetResourceReplaced

Autogenerated Proxy Method

Audio (`Neos\Media\Domain\Model\Audio`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

Document (`Neos\Media\Domain\Model\Document`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

Image (`Neos\Media\Domain\Model\Image`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

ImageVariant (`Neos\Media\Domain\Model\ImageVariant`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

Thumbnail (`Neos\Media\Domain\Model\Thumbnail`)

This class contains the following signals.

thumbnailCreated

Autogenerated Proxy Method

ThumbnailService (`Neos\Media\Domain\Service\ThumbnailService`)

This class contains the following signals.

thumbnailCreated

Autogenerated Proxy Method

Video (`Neos\Media\Domain\Model\Video`)

This class contains the following signals.

assetCreated

Autogenerated Proxy Method

6.8.4 Neos Signals Reference

This reference was automatically generated from code on 2019-03-05

AbstractCreate (`Neos\Neos\Ui\Domain\Model\Changes\AbstractCreate`)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

ContentContext (`Neos\Neos\Domain\Service\ContentContext`)

This class contains the following signals.

beforeAdoptNode

Autogenerated Proxy Method

afterAdoptNode

Autogenerated Proxy Method

ContentController (`Neos\Neos\Controller\Backend\ContentController`)

This class contains the following signals.

assetUploaded

Autogenerated Proxy Method

Create (`Neos\Neos\Ui\Domain\Model\Changes\Create`)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

CreateAfter (`Neos\Neos\Ui\Domain\Model\Changes\CreateAfter`)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

CreateBefore (`Neos\Neos\Ui\Domain\Model\Changes\CreateBefore`)

This class contains the following signals.

nodeCreationHandlersApplied

Autogenerated Proxy Method

PublishingService (`Neos\Neos\Service\PublishingService`)

This class contains the following signals.

nodePublished

Autogenerated Proxy Method

nodeDiscarded

Autogenerated Proxy Method

Site (`Neos\Neos\Domain\Model\Site`)

This class contains the following signals.

siteChanged

Autogenerated Proxy Method

SiteImportService (`Neos\Neos\Domain\Service\SiteImportService`)

This class contains the following signals.

siteImported

Autogenerated Proxy Method

SiteService (`Neos\Neos\Domain\Service\SiteService`)

This class contains the following signals.

sitePruned

Autogenerated Proxy Method

UserService (`Neos\Neos\Domain\Service\UserService`)

This class contains the following signals.

userCreated

Autogenerated Proxy Method

userDeleted

Autogenerated Proxy Method

userUpdated

Autogenerated Proxy Method

rolesAdded

Autogenerated Proxy Method

rolesRemoved

Autogenerated Proxy Method

userActivated

Autogenerated Proxy Method

userDeactivated

Autogenerated Proxy Method

6.9 Coding Guideline Reference

6.9.1 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\ControllerInterface	ControllerInterface	Consequently the interface belongs to all the controllers in the Controller sub namespace
\Neos\Flow\Mvc\Controller\ControllerInterface		Better
\Neos\Cache\AbstractBackend	AbstractBackend	Same here: In reality this class belongs to the backends
\Neos\Cache\Backend\AbstractBackend		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
```

(continues on next page)

(continued from previous page)

```
                                                    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() {
 ...
}
```

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.

---

[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.

6.9.2 JavaScript Coding Guidelines

Here, you will find an explanation of the JavaScript Coding Guidelines we use. Generally, we strive to follow the Neos Flow Coding Guidelines as closely as possible, with exceptions which make sense in the JavaScript context.

This guideline explains mostly how we want JavaScript code to be formatted; and it does **not** deal with the Neos User Interface structure. If you want to know more about the Neos User Interface architecture, have a look into the "Neos User Interface Development" book.

Naming Conventions

- one class per file, with the same naming convention as Neos Flow.
- This means all classes are built like this: `<PackageKey>.<SubNamespace>.<ClassName>`, and this class is implemented in a JavaScript file located at `<Package>/.../JavaScript/<SubNamespace>/<ClassName>.js`
- Right now, the base directory for JavaScript in Neos Flow packages `Resources/Public/JavaScript`, but this might still change.
- We suggest that the base directory for JavaScript files is *JavaScript*.
- Files have to be encoded in UTF-8 without byte order mark (BOM).
- Classes and namespaces are written in `UpperCamelCase`, while properties and methods are written in `lowerCamelCase`.
- The xtype of a class is always the fully qualified class name. Every class which can be instantiated needs to have an xtype declaration.
- Never create a class which has classes inside itself. Example: if the class `TYPO3.Foo` exists, it is prohibited to create a class `TYPO3.Foo.Bar`.You can easily check this: If a directory with the same name as the JavaScript file exists, this is prohibited.

  Here follows an example:

```
TYPO3.Foo.Bar // implemented in .../Foo/Bar.js
TYPO3.Foo.Bar = ...

TYPO3.Foo // implemented in ...Foo.js
TYPO3.Foo = ..... **overriding the "Bar" class**
```

  So, if the class `TYPO3.Foo.Bar` is included **before** `TYPO3.Foo`, then the second class definition completely overrides the `Bar` object. In order to prevent such issues, this constellation is forbidden.
- Every class, method and class property should have a doc comment.
- Private methods and properties should start with an underscore (_) and have a `@private` annotation.

Doc Comments

Generally, doc comments follow the following form:

```
/**
 *
 */
```

See the sections below on which doc comments are available for the different elements (classes, methods, ...).

We are using http://code.google.com/p/ext-doc/ for rendering an API documentation from the code, that's why types inside `@param`, `@type` and `@cfg` have to be written in braces like this:

```
@param {String} theFirstParameter A Description of the first parameter
@param {My.Class.Name} theSecondParameter A description of the second parameter
```

Generally, we do not use `@api` annotations, as private methods and attributes are marked with `@private` and prefixed with an underscore. So, **everything which is not marked as private belongs to the public API!**

We are not sure yet if we should use `@author` annotations at all. (TODO Decide!)

To make a reference to another method of a class, use the `{@link #methodOne This is an example link to method one}` syntax.

If you want to do multi-line doc comments, you need to format them with `<br>`, `<pre>` and other HTML tags:

```
/**
 * Description of the class. Make it as long as needed,
 * feel free to explain how to use it.
 * This is a sample class <br/>
 * The file encoding should be utf-8 <br/>
 * UTF-8 Check: öäüß <br/>
 * {@link #methodOne This is an example link to method one}
 */
```

Class Definitions

Classes can be declared singleton or prototype. A class is **singleton**, if only one instance of this class will exist at any given time. An class is of type **prototype**, if more than one object can be created from the class at run-time. Most classes will be of type **prototype**.

You will find examples for both below.

Prototype Class Definitions

Example of a prototype class definition:

```
Ext.ns("TYPO3.TYPO3.Content");


/*
 * This file is part of the Neos.Neos package.
 *
 * (c) Contributors of the Neos Project - www.neos.io
 *
 * This package is Open Source Software. For the full copyright and license
```

(continues on next page)

(continued from previous page)

```
 * information, please view the LICENSE file which was distributed with this
 * source code.
 */

/**
 * @class TYPO3.TYPO3.Content.FrontendEditor
 *
 * The main frontend editor.
 *
 * @namespace TYPO3.TYPO3.Content
 * @extends Ext.Container
 */
TYPO3.TYPO3.Content.FrontendEditor = Ext.extend(Ext.Container, {
        // here comes the class contents
});
Ext.reg('TYPO3.TYPO3.Content.FrontendEditor', TYPO3.TYPO3.Content.FrontendEditor);
```

- At the very beginning of the file is the namespace declaration of the class, followed by a newline.
- Then follows the class documentation block, which **must** start with the `@class` declaration in the first line.
- Now comes a description of the class, possibly with examples.
- Afterwards **must** follow the namespace of the class and the information about object extension
- Now comes the actual class definition, using `Ext.extend`.
- As the last line of the class, it follows the xType registration. We always use the fully qualified class name as xtype

Usually, the constructor of the class receives a hash of parameters. The possible configuration options need to be documented inside the class with the `@cfg` annotation:

```
TYPO3.TYPO3.Content.FrontendEditor = Ext.extend(Ext.Container, {
        /**
         * An explanation of the configuration option followed
         * by a blank line.
         *
         * @cfg {Number} configTwo
         */
        configTwo: 10
        ...
}
```

Singleton Class Definitions

Now comes a singleton class definition. You will see that it is very similar to a prototype class definition, we will only highlight the differences.

Example of a singleton class definition:

```
Ext.ns("TYPO3.TYPO3.Core");


/*
 * This file is part of the Neos.Neos package.
 *
```

(continues on next page)

(continued from previous page)

```
 * (c) Contributors of the Neos Project - www.neos.io
 *
 * 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.
 */

/**
 * @class TYPO3.TYPO3.Core.Application
 *
 * The main entry point which controls the lifecycle of the application.
 *
 * @namespace TYPO3.TYPO3.Core
 * @extends Ext.util.Observable
 * @singleton
 */
TYPO3.TYPO3.Core.Application = Ext.apply(new Ext.util.Observable, {
        // here comes the class contents
});
```

- You should add a `@singleton` annotation to the class doc comment after the `@namespace` and `@extends` annotation
- In singleton classes, you use `Ext.apply`. Note that you need to use `new` to instantiate the base class.
- There is **no xType** registration in singletons, as they are available globally anyhow.

Class Doc Comments

Class Doc Comments should always be in the following order:

- `@class <Name.Of.Class>` (required)
- Then follows a description of the class, which can span multiple lines. Before and after this description should be a blank line.
- `@namespace <Name.Of.Namespace>` (required)
- `@extends <Name.Of.BaseClass>` (required)
- `@singleton` (required if the class is a singleton)

If the class has a non-empty constructor, the following doc comments need to be added as well, after a blank line:

- `@constructor`
- `@param {<type>} <nameOfParameter> <description of parameter>` for every parameter of the constructor

Example of a class doc comment without constructor:

```
/**
 * @class Acme.Foo.Bar
 *
 * Some Description of the class,
 * which can possibly span multiple lines
 *
 * @namespace Acme.Foo
```

(continues on next page)

(continued from previous page)

```
 * @extends TYPO3.TYPO3.Core.SomeOtherClass
 */
```

Example of a class doc comment with constructor:

```
/**
 * @class Acme.TYPO3.Foo.ClassWithConstructor
 *
 * This class has a constructor!
 *
 * @namespace Acme.TYPO3.Foo
 * @extends TYPO3.TYPO3.Core.SomeOtherClass
 *
 * @constructor
 * @param {String} id The ID which to use
 */
```

Method Definitions

Methods should be documented the following way, with a blank line between methods.

Example of a method comment:

```
...
TYPO3.TYPO3.Core.Application = Ext.apply(new Ext.util.Observable, {
        ... property definitions ...
        /**
         * This is a method declaration; and the
         * explanatory text is followed by a newline.
         *
         * @param {String} param1 Parameter name
         * @param {String} param2 (Optional) Optional parameter
         * @return {Boolean} Return value
         */
        aPublicMethod: function(param1, param2) {
                return true;
        },

        /**
         * this is a private method of this class,
         * the private annotation marks them an prevent that they
         * are listed in the api doc. As they are private, they
         * have to start with an underscore as well.
         *
         * @return {void}
         * @private
         */
        _sampleMethod: function() {
        }
}
...
```

Contrary to what is defined in the Neos Flow PHP Coding Guidelines, methods which are public **automatically belong to the public API**, without an `@api` annotation. Contrary, methods which do **not belong to the public API** need to begin with an underscore and have the `@private` annotation.

- All methods need to have JSDoc annotations.
- Every method needs to have a `@return` annotation. In case the method does not return anything, a `@return {void}` is needed, otherwise the concrete return value should be described.

Property Definitions

All properties of a class need to be properly documented as well, with an `@type` annotation. If a property is private, it should start with an underscore and have the `@private` annotation at the last line of its doc comment:

```
...
TYPO3.TYPO3.Core.Application = Ext.apply(new Ext.util.Observable, { // this is just␣
↪an example class definition
        /**
         * Explanation of the property
         * which is followed by a newline
         *
         * @type {String}
         */
        propertyOne: 'Hello',

        /**
         * Now follows a private property
         * which starts with an underscore.
         *
         * @type {Number}
         * @private
         */
        _thePrivateProperty: null,
        ...
}
```

Code Style

- use single quotes(') instead of double quotes(") for string quoting
- Multi-line strings (using \) are forbidden. Instead, multi-line strings should be written like this:

```
'Some String' +
' which spans' +
' multiple lines'
```

- There is no limitation on line length.
- JavaScript constants (true, false, null) must be written in lowercase, and not uppercase.
- Custom JavaScript constants should be avoided.
- Use a single `var` statement at the top of a method to declare all variables:

```
function() {
        var myVariable1, myVariable2, someText;
        // now, use myVariable1, ....
}

Please do **not assign** values to the variables in the initialization, except␣
↪empty
```

(continues on next page)

(continued from previous page)

```
default values::

// DO:
function() {
        var myVariable1, myVariable2;
        ...
}
// DO:
function() {
        var myVariable1 = {}, myVariable2 = [], myVariable3;
        ...
}
// DON'T
function() {
        var variable1 = 'Hello',
                variable2 = variable1 + ' World';
        ...
}
```

- We use **a single TAB** for indentation.
- Use inline comments sparingly, they are often a hint that a new method must be introduced.

  Inline Comments must be indented **one level deeper** than the current nesting level:

```
function() {
        var foo;
                // Explain what we are doing here.
        foo = '123';
}
```

- Whitespace around control structures like `if`, `else`, ... should be inserted like in the Neos Flow CGLs:

```
if (myExpression) {
        // if part
} else {
        // Else Part
}
```

- Arrays and Objects should **never** have a trailing comma after their last element
- Arrays and objects should be formatted in the following way:

```
[
        {
                foo: 'bar'
        }, {
                x: y
        }
]
```

- Method calls should be formatted the following way:

```
// for simple parameters:
new Ext.blah(options, scope, foo);
object.myMethod(foo, bar, baz);

// when the method takes a **single** parameter of type **object** as argument,␣
↪and this object is specified directly in place:
```

(continues on next page)

(continued from previous page)

```
new Ext.Panel({
        a: 'b',
        c: 'd'
});

// when the method takes more parameters, and one is a configuration object which␣
↪is specified in place:
new Ext.blah(
        {
                foo: 'bar'
        },
        scope,
        options
);<
```

TODO: are there JS Code Formatters / Indenters, maybe the Spket JS Code Formatter?

Using JSLint to validate your JavaScript

JSLint is a JavaScript program that looks for problems in JavaScript programs. It is a code quality tool. When C was a young programming language, there were several common programming errors that were not caught by the primitive compilers, so an accessory program called `lint` was developed that would scan a source file, looking for problems. `jslint` is the same for JavaScript.

JavaScript code ca be validated on-line at http://www.jslint.com/. When validating the JavaScript code, "The Good Parts" family options should be set. For that purpose, there is a button "The Good Parts" to be clicked.

Instead of using it online, you can also use JSLint locally, which is now described. For the sake of convenience, the small tutorial bellow demonstrates how to use JSlint with the help of CLI wrapper to enable recursive validation among directories which streamlines the validation process.

- Download Rhino from http://www.mozilla.org/rhino/download.html and put it for instance into `/Users/john/WebTools/Rhino`
- Download `JSLint.js` (@see attachment "jslint.js", line 5667-5669 contains the configuration we would like to have, still to decide) (TODO)
- Download `jslint.php` (@see attachment "jslint.php" TODO), for example into `/Users/fudriot/WebTools/JSLint`
- Open and edit path in `jslint.php` -> check variable `$rhinoPath` and `$jslintPath`
- Add an alias to make it more convenient in the terminal:

```
alias jslint '/Users/fudriot/WebTools/JSLint/jslint.php'
```

Now, you can use JSLint locally:

```
// scan one file or multi-files
jslint file.js
jslint file-1.js file-2.js

// scan one directory or multi-directory
jslint directory
jslint directory-1 directory-2

// scan current directory
jslint .
```

It is also possible to adjust the validation rules JSLint uses. At the end of file `jslint.js`, it is possible to customize the rules to be checked by JSlint by changing options' value. By default, the options are taken over the book "JavaScript: The Good Parts" which is written by the same author of JSlint.

Below are the options we use for TYPO3 v5:

```
bitwise: true, eqeqeq: true, immed: true,newcap: true, nomen: false,
onevar: true, plusplus: false, regexp: true, rhino: true, undef: false,
white: false, strict: true
```

In case some files needs to be evaluated with special rules, it is possible to add a comment on the top of file which can override the default ones:

```
/* jslint white: true, evil: true, laxbreak: true, onevar: true, undef: true,
nomen: true, eqeqeq: true, plusplus: true, bitwise: true, regexp: true,
newcap: true, immed: true */
```

More information about the meaning and the reasons of the rules can be found at http://www.jslint.com/lint.html

Event Handling

When registering an event handler, always use explicit functions instead of inline functions to allow overriding of the event handler.

Additionally, this function needs to be prefixed with `on` to mark it as event handler function. Below follows an example for good and bad code.

Good Event Handler Code:

```
TYPO3.TYPO3.Application.on('theEventName', this._onCustomEvent, this);
```

Bad Event Handler Code:

```
TYPO3.TYPO3.Application.on(
        'theEventName',
        function() {
                alert('Text');
        },
        this
);
```

All events need to be explicitly documented inside the class where they are fired onto with an `@event` annotation:

```
TYPO3.TYPO3.Core.Application = Ext.apply(new Ext.util.Observable, {
        /**
         * @event eventOne Event declaration
         */

        /**
         * @event eventTwo Event with parameters
         * @param {String} param1 Parameter name
         * @param {Object} param2 Parameter name
         * <ul>
         * <li><b>property1:</b> description of property1</li>
         * <li><b>property2:</b> description of property2</li>
         * </ul>
         */
```

(continues on next page)

(continued from previous page)

```
        ...
}
```

Additionally, make sure to document if the scope of the event handler is not set to `this`, i.e. does not point to its class, as the user expects this.

ExtJS specific things

TODO

- explain initializeObject
- how to extend Ext components
- can be extended by using constructor() not initComponents() like it is for panels and so on

How to extend data stores

This is an example for how to extend an ExtJS data store:

```
TYPO3.TYPO3.Content.DummyStore = Ext.extend(Ext.data.Store, {

        constructor: function(cfg) {
                cfg = cfg || {};
                var config = Ext.apply(
                        {
                                autoLoad: true
                        },
                        cfg
                );

                TYPO3.TYPO3.Content.DummyStore.superclass.constructor.call(
                        this,
                        config
                );
        }
});
Ext.reg('TYPO3.TYPO3.Content.DummyStore', TYPO3.TYPO3.Content.DummyStore);
```

Unit Testing

- It's highly recommended to write unit tests for javascript classes. Unit tests should be located in the following location: `Package/Tests/JavaScript/...`
- The structure below this folder should reflect the structure below `Package/Resources/Public/JavaScript/...` if possible.
- The namespace for the Unit test classes is `Package.Tests`.
- TODO: Add some more information about Unit Testing for JS
- TODO: Add note about the testrunner when it's added to the package
- TODO: http://developer.yahoo.com/yui/3/test/

Note: This is a documentation stub.

6.10 Configuration Reference

6.10.1 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
```

6.10.2 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'
```

(continues on next page)

(continued from previous page)

```
            icon: 'icon-gavel'
          baseNodeType: 'Acme.Com:Document.Imprint,Acme.Com:Document.Terms'
        'landingPages':
          ui:
            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.

6.11 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.

6.11.1 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`
- `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

---

6.11.2 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.

6.11.3 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.

Workspace

Filter nodes by workspace name.

Options Reference:

`workspaceName` (string) The workspace name to match on.

CHAPTER

SEVEN

CONTRIBUTE

7.1 Development

Developing Neos.

7.1.1 Neos UI Development

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.

7.2 Documentation

Improving the Neos documentation.

7.2.1 Neos Documentation

How it works

We use Read The Docs (http://neos.readthedocs.org) to host the 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.

7.2.2 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]

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)

[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.

- 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:

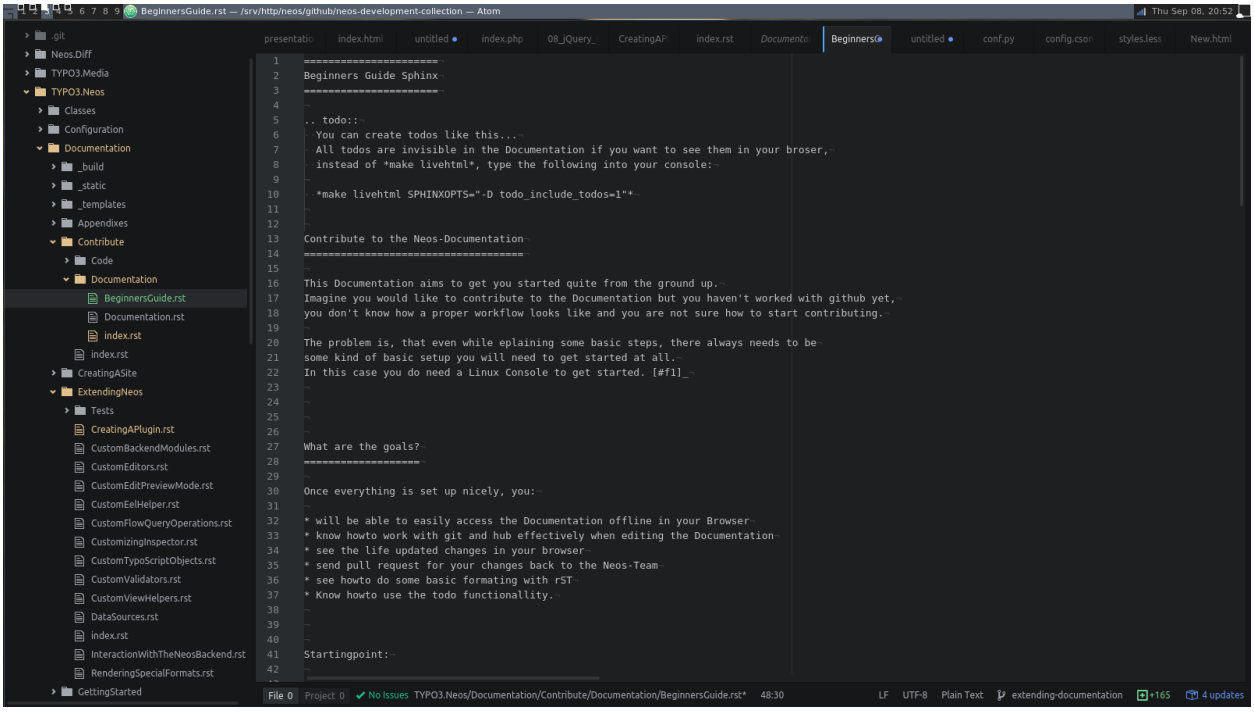


Fig. 1: The Atom Editor

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)
```

[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. . .

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
```

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

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.

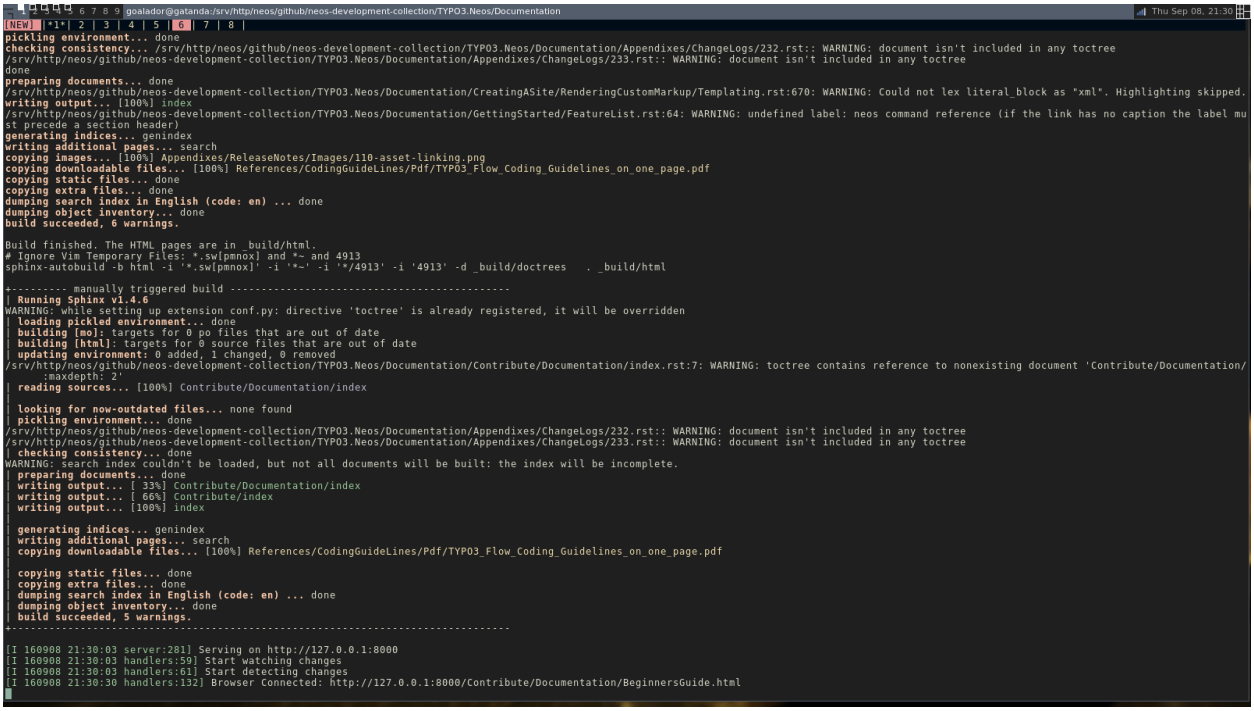


Fig. 2: Sphinx make livehtml

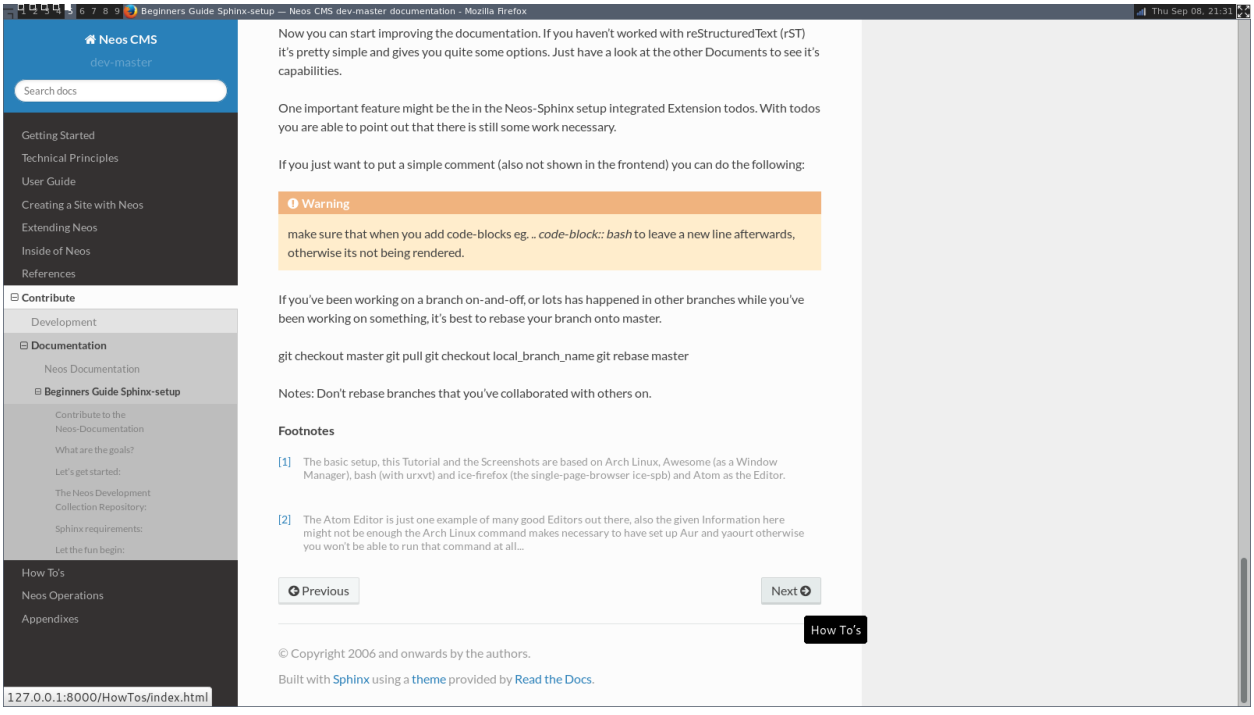


Fig. 3: Sphinx browser view

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"
```

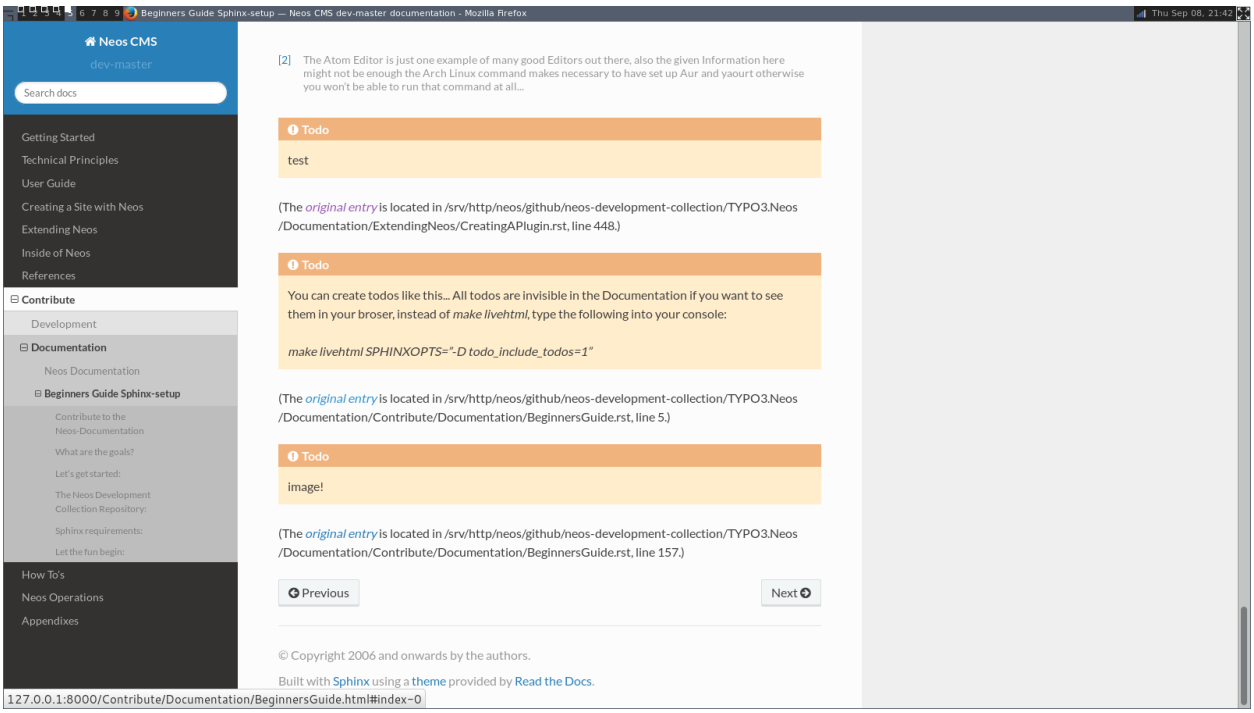


Fig. 4: Sphinx todolist

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.

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

EIGHT

HOW TO'S

8.1 Neos Best Practices (to be written)

8.2 Adding A Simple Contact Form

Using the Neos.Form package you can easily create and adopt simple to very complex forms. For it to work properly you just have to define where it should find its form configurations.

Yaml (Sites/Vendor.Site/Configuration/Settings.yaml)

```
Neos:
  Form:
    yamlPersistenceManager:
      savePath: 'resource://Vendor.Site/Private/Form/'
```

Now place a valid Neos.Form Yaml configuration in the Private/Form folder. Then add a Form Element where you wish the form to be displayed and select it from the dropdown in the Inspector.

Yaml (Sites/Vendor.Site/Resources/Private/Form/contact-form.yaml)

```
type: 'Neos.Form:Form'
identifier: contact-form
label: Contact
renderingOptions:
  submitButtonLabel: Send
renderables:
  -
    type: 'Neos.Form:Page'
    identifier: page-one
    label: Contact
    renderables:
      -
        type: 'Neos.Form:SingleLineText'
        identifier: name
        label: Name
        validators:
          - identifier: 'Neos.Flow:NotEmpty'
        properties:
          placeholder: Name
        defaultValue: ''
      -
        type: 'Neos.Form:SingleLineText'
        identifier: email
```

(continues on next page)

(continued from previous page)

```
        label: E-Mail
        validators:
          - identifier: 'Neos.Flow:NotEmpty'
          - identifier: 'Neos.Flow:EmailAddress'
        properties:
          placeholder: 'E-Mail'
        defaultValue: ''
      -
        type: 'Neos.Form:MultiLineText'
        identifier: message
        label: Message
        validators:
          - identifier: 'Neos.Flow:NotEmpty'
        properties:
          placeholder: 'Your Message'
        defaultValue: ''
finishers:
  -
    identifier: 'Neos.Form:Email'
    options:
      templatePathAndFilename: resource://Vendor.Site/Private/Templates/Email/Message.
↪txt
      subject: Contact from example.net
      recipientAddress: office@example.net
      recipientName: 'Office of Company'
      senderAddress: server@example.net
      senderName: Server example.net
      replyToAddress: office@example.net
      format: plaintext
  -
    identifier: 'Neos.Form:Confirmation'
    options:
      message: >
        <h3>Thank you for your feedback</h3>
        <p>We will process it as soon as possible.</p>
```

In this example we are using the Neos.Form:Email Finisher. The Email Finisher requires the Neos.SwiftMailer package to be installed. It sends an E-Mail using the defined template and settings. By the second Finisher a confirmation is displayed.

Html (Sites/Vendor.Site/Resources/Private/Templates/Email/Message.txt)

```
Hello,

<f:for each="{form.formState.formValues}" as="value" key="label">
  {label}: {value}
</f:for>

Thanks
```

To find out more about how to create forms see the Neos.Form package. There is even a Click Form Builder that exports the Yaml settings files.

Warning: Make sure the Neos.Demo package (or other) is deactivated. Otherwise the setting `Neos.Form.yamlPersistenceManager.savePath` may be overwritten by another package. You can deactivate a pack-

age with the command `./flow package:deactivate <PackageKey>`.

8.3 Changing the Body Class with a condition

In some cases there is the need to define different body classes based on a certain condition.

It can for example be that if a page has sub pages then we want to add a body class tag for this.

Fusion code:

```
page {
    bodyTag {
        attributes.class = ${q(node).children().count() > 1 ? 'has-subpages' : ''}
    }
}
```

First of all we add the part called *bodyTag* to the Fusion page object. Then inside we add the *attributes.class*.

Then we add a FlowQuery that checks if the current node has any children. If the condition is true then the class "has-subpages" is added to the body tag on all pages that have any children.

An other example could be that we want to check if the current page is of type page.

Fusion code:

```
page {
    bodyTag {
        attributes.class = ${q(node).filter('[instanceof Neos.Neos:Page]') != '' ?
↪'is-page' : ''}
    }
}
```

8.4 Changing Defaults Depending on Content Placement

Let's say we want to adjust our *YouTube* content element depending on the context: By default, it renders in a standard YouTube video size; but when being used inside the sidebar of the page, it should shrink to a width of 200 pixels. This is possible through *nested prototypes*:

```
page.body.contentCollections.sidebar.prototype(My.Package:YouTube) {
  width = '200'
  height = '150'
}
```

Essentially the above code can be read as: "For all YouTube elements inside the sidebar of the page, set width and height".

Let's say we also want to adjust the size of the YouTube video when being used in a *ThreeColumn* element. This time, we cannot make any assumptions about a fixed Fusion path being rendered, because the *ThreeColumn* element can appear both in the main column, in the sidebar and nested inside itself. However, we are able to *nest prototypes into each other*:

```
prototype(ThreeColumn).prototype(My.Package:YouTube) {
  width = '200'
  height = '150'
}
```

This essentially means: “For all YouTube elements which are inside ThreeColumn elements, set width and height”.

The two possibilities above can also be flexibly combined. Basically this composability allows to adjust the rendering of websites and web applications very easily, without overriding templates completely.

After you have now had a head-first start into Fusion based on practical examples, it is now time to step back a bit, and explain the internals of Fusion and why it has been built this way.

8.5 Creating a simple Content Element

If you need some specific content element, you can easly create a new Node Type with an attached HTML template. To add a new Node Type, follow this example, just replace “Vendor” by your own vendor prefix:

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Vendor:YourContentElementName':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    label: 'My first custom content element'
    group: 'general'
    inspector:
      groups:
        image:
          label: 'Image'
          icon: 'icon-image'
          position: 1
  properties:
    headline:
      type: string
      defaultValue: 'Replace by your headline value ...'
      ui:
        label: 'Headline'
        inlineEditable: TRUE
    subheadline:
      type: string
      defaultValue: 'Replace by your subheadline value ...'
      ui:
        label: 'Subheadline'
        inlineEditable: TRUE
    text:
      type: string
      ui:
        label: 'Text'
        reloadIfChanged: TRUE
    image:
      type: Neos\Media\Domain\Model\ImageInterface
      ui:
        label: 'Image'
        reloadIfChanged: TRUE
```

(continues on next page)

(continued from previous page)

```
        inspector:
          group: 'image'
```

Based on your Node Type configuration, now you need a Fusion object to be able to use your new Node Type. This Fusion object needs to have the same name as the Node Type:

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/Root.fusion):

```
prototype(Vendor:YourContentElementName) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://Vendor.Site/Private/Templates/FusionObjects/
↪YourContentElementName.html'

        headline = ${q(node).property('headline')}
        subheadline = ${q(node).property('subheadline')}
        text = ${q(node).property('text')}
        image = ${q(node).property('image')}
}
```

Last thing, add the required Fluid template:

HTML (Vendor.Site/Private/Templates/FusionObjects/YourContentElementName.html):

```
{namespace neos=Neos\Neos\ViewHelpers}
{namespace media=Neos\Media\ViewHelpers}
<article>
        <header>
                {neos:contentElement.editable(property: 'headline', tag: 'h2')}
                {neos:contentElement.editable(property: 'subheadline', tag: 'h3')}
        </header>
        <div>
                {neos:contentElement.editable(property: 'text')}
                <f:if condition="{image}"><media:image image="{image}" maximumWidth=
↪"300" alt="{headline}" /></f:if>
        </div>
</article>
```

Now, if you try to add a new Node in your page, you should see your new Node Type. Enjoy editing with Neos.

8.6 Customize Login Screen

You can customize the login screen by editing your `Settings.yaml`:

```
Neos:
  Neos:
    userInterface:
      backendLoginForm:
        backgroundImage: 'resource://Your.Package/Public/Images/LoginScreen.jpg'
```

Or alternatively add a custom stylesheet:

```
Neos:
  Neos:
    userInterface:
      backendLoginForm:
```

(continues on next page)

(continued from previous page)

```
        stylesheets:
          'Your.Package:CustomStyles': 'resource://Your.Package/Public/Styles/Login.
↪css'
```

Note: In this case `Your.Package:CustomStyles` is a simple key, used only internally.

8.6.1 How to disable a stylesheet ?

You can disable existing stylesheets, by setting the value to `FALSE`, the following snippet will disable the stylesheet provided by Neos, so your are free to implement your own:

```
Neos:
  Neos:
    userInterface:
      backendLoginForm:
        stylesheets:
          'Neos.Neos:DefaultStyles': FALSE
          'Your.Package:CustomStyles': 'resource://Your.Package/Public/Styles/Login.
↪css'
```

8.7 Editing a shared footer across all pages

A shared footer in Neos works as follows:

- The homepage contains a collection of content elements
- The same collection is rendered on all other pages

This enables you to edit the footer on all pages.

To add the footer to the page you use the *ContentCollection* with a static node path.

To have the collection on the homepage you need to configure the childNodes structure of the homepage. For this you create a homepage node type with for example the following configuration in NodeTypes.yaml:

```
'My.Package:HomePage':
  superTypes:
    'Neos.NodeTypes:Page': TRUE
  ui:
    label: 'Homepage'
  childNodes:
    footer:
      type: 'Neos.Neos:ContentCollection'
```

Note: If you run into the situation that the child nodes for your page are missing (for example if you manually updated the node type in the database) you might have to create the missing child nodes using:

```
./flow node:repair --node-type Neos.NodeTypes:Page
```

Fusion code:

```
footer = Neos.Neos:ContentCollection {
        nodePath = ${q(site).find('footer').property('_path')}
        collection = ${q(site).children('footer').children()}
}
```

Of course you have to update the selection in the example if your footer is not stored on the site root, but for example on a page named 'my-page'. The selection would then be: *${q(site).find('my-page').children('footer').children()}*.

8.8 Extending the Page

In Neos the page is a simple Node Type named Neos.Neos:Page, you can directly extend this Node Type to add specific properties. Below you will find a simple example for adding a page background image:

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml)

```
'Neos.NodeTypes:Page':
  ui:
    inspector:
      groups:
        background:
          label: 'Background'
          position: 900
  properties:
    backgroundImage:
      type: Neos\Media\Domain\Model\ImageInterface
      ui:
        label: 'Image'
        reloadPageIfChanged: TRUE
        inspector:
          group: 'background'
```

With this configuration, when you click on the page, you will see the Image editor in the Inspector.

To access the backgroundImage in your page template you can also modify the Neos.Neos:Page Fusion object, like in the below example:

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/Root.fusion)

```
prototype(Neos.Neos:Page) {
        body.backgroundImage = ${q(node).property('backgroundImage')}
}
```

With Neos.Media ViewHelper you can display the Image with the follwing HTML snippet:

HTML

```
{namespace media=Neos\Media\ViewHelpers}
<style>
html {
        margin:0;
        padding:0;
        background: url({media:uri.image(image:backgroundImage)}) no-repeat center␣
↪fixed;
        -webkit-background-size: cover;
        -moz-background-size: cover;
        -o-background-size: cover;
```

(continues on next page)

(continued from previous page)

```
        background-size: cover;
}
</style>
```

8.9 Integrating a JavaScript-based slider

If you want to integrate a Slider into your page as content element or as part of your template and want edit it in the backend you have do some simple steps.

First you have to use a slider javscript plugin which initializes itself when added to the page after page load. Or you write your own initialization code into a javascript function which you then add as callback for the neos backend events.

For this example the carousel plugin and styling from bootstrap 3.0 has been used: http://getbootstrap.com/javascript/#carousel

To create the basic content element you have to add it to your node types.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml):

```
'Vendor.Site:Carousel':
  superTypes:
    'Neos.Neos:Content': TRUE
  childNodes:
    carouselItems:
      type: 'Neos.Neos:ContentCollection'
  ui:
    label: 'Carousel'
    group: 'plugins'
    icon: 'icon-picture'
    inlineEditable: TRUE
```

Next you need to define the prototype for the slider in typoscript.

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/NodeTypes/Carousel.fusion):

```
prototype(Vendor.Site:Carousel) {
        carouselItems = Neos.Neos:ContentCollection {
                nodePath = 'carouselItems'
                content.iterationName = 'carouselItemsIteration'
                attributes.class = 'carousel-inner'
        }

        // Collect the carousels children but only images
        carouselItemArray = ${q(node).children('carouselItems').children('[instanceof␣
↪Neos.NodeTypes:Image]')}

        // Enhance image prototype when inside the carousel
        prototype(Neos.NodeTypes:Image) {
                // Render images in the carousel with a special template.
                templatePath = 'resource://Vendor.Site/Private/Templates/
↪FusionObjects/CarouselItem.html'

                // The first item should later be marked as active
                attributes.class = ${'item' + (carouselItemsIteration.isFirst ? '␣
↪active' : '')}
```

(continues on next page)

(continued from previous page)

```
                // We want to use the item iterator in fluid so we have to store it
↪as variable.
                iteration = ${carouselItemsIteration}
        }
}
```

Now you need to include this at the top of your (Sites/Vendor.Site/Resources/Private/Fusion/Root.fusion):

```
// Includes all additional ts2 files inside the NodeTypes folder
include: NodeTypes/*.fusion
```

For rendering you need the fluid templates for the slider.

Html (Sites/Vendor.Site/Private/Templates/NodeTypes/Carousel.html)

```
{namespace neos=Neos\Neos\ViewHelpers}
{namespace fusion=Neos\Fusion\ViewHelpers}
<div{attributes -> f:format.raw()}>
        <div class="carousel slide" id="{node.identifier}">
                <!-- Indicators -->
                <ol class="carousel-indicators">
                        <f:for each="{carouselItemArray}" as="item" iteration=
↪"itemIterator">
                                <li data-target="#{node.identifier}" data-slide-to="
↪{itemIterator.index}" class="{f:if(condition: itemIterator.isFirst, then: 'active')}
↪"></li>
                        </f:for>
                </ol>

                <!-- Wrapper for slides -->
                {carouselItems -> f:format.raw()}

                <!-- Controls -->
                <a class="left carousel-control" href="#{node.identifier}" data-slide=
↪"prev">
                        <span class="icon-prev"></span>
                </a>
                <a class="right carousel-control" href="#{node.identifier}" data-
↪slide="next">
                        <span class="icon-next"></span>
                </a>
        </div>
</div>
```

And now the fluid template for the slider items.

Html (Sites/Vendor.Site/Private/Templates/FusionObjects/CarouselItem.html)

```
{namespace neos=Neos\Neos\ViewHelpers}
{namespace media=Neos\Media\ViewHelpers}
<div{attributes -> f:format.raw()}>
        <f:if condition="{image}">
                <f:then>
                        <media:image image="{image}" alt="{alternativeText}" title="
↪{title}" maximumWidth="{maximumWidth}" maximumHeight="{maximumHeight}" />
                </f:then>
```

(continues on next page)

(continued from previous page)

```
                <f:else>
                        <img src="{f:uri.resource(package: 'Neos.Neos', path: 'Images/
↪dummy-image.svg')}" title="Dummy image" alt="Dummy image" />
                </f:else>
        </f:if>
        <div class="carousel-caption">
                <f:if condition="{hasCaption}">
                        {neos:contentElement.editable(property: 'caption')}
                </f:if>
        </div>
</div>
```

For styling you can simply include the styles provided in bootstrap into your page template.

Html

```
<link rel="stylesheet" href="{f:uri.resource(path: '3/css/bootstrap.min.css',␣
↪package: 'Neos.Twitter.Bootstrap')}" media="all" />
```

If you want to hide specific parts of a plugin while in backend you can use the provided neos-backend class.

Css

```
.neos-backend .carousel-control {
        display: none;
}
```

Don't forget to include the javascript for the plugin from the bootstrap package into your page template.

Html

```
<script src="{f:uri.resource(path: '3/js/bootstrap.min.js', package: 'Neos.Twitter.
↪Bootstrap')}"></script>
```

Now, you should be able to add the new 'Carousel' node type as content element.

8.10 Rendering Custom Document Types

8.10.1 Select Template based on NodeType

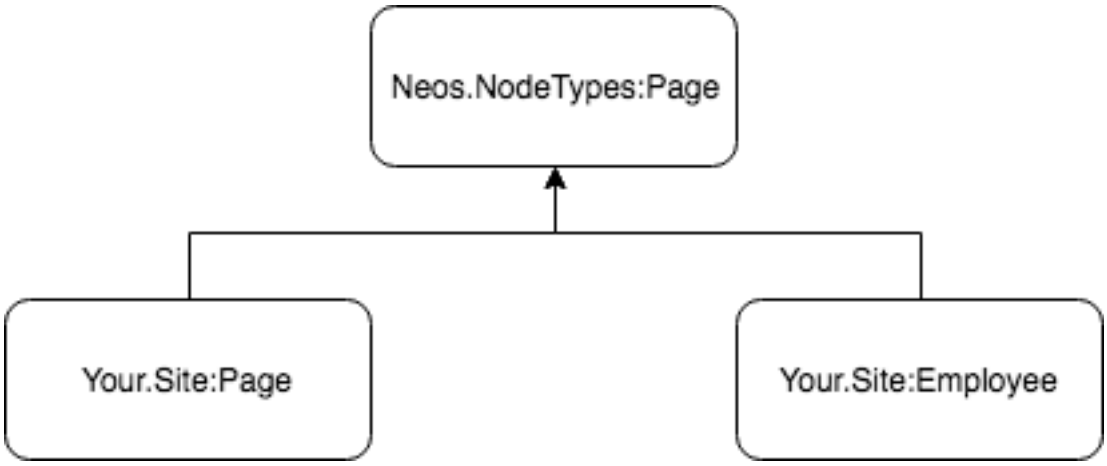
It is possible to select the page rendering configuration based on the node type of the page. Let's say you have a custom node type named `Your.Site:Page` which has `Neos.NodeTypes:Page` as a supertype. You added a `Your.Site:Employee` page which is used for displaying a personal page of employees working in your company. This page should have a different rendering output compared to your basic page.

The right approach would be to create a Fusion prototype for your default page and employee page like:

```
prototype(Your.Site:Page) < prototype(Neos.Neos:Page) {
    body.templatePath = 'resource://Your.Site/Private/Templates/Page/Default.html'
    # Your further page configuration here
}

prototype(Your.Site:EmployeePage) < prototype(Your.Site:Page) {
    body.templatePath = 'resource://Your.Site/Private/Templates/Page/Employee.html'
```

(continues on next page)



(continued from previous page)

```
    # Your further employee page configuration here
}
```

Because Neos provides the documentType matcher out of the box (see *Rendering A Page*), these prototypes will be automatically picked up and rendered by Fusion, giving you the possibility to control the rendering for each page type individually.

8.11 Rendering a Menu

The implementation of a menu is done in Fusion and HTML, this gives an flexibility in what can be rendered.

First of all you have to add a new element (with a name) in Fusion that is of type Menu. Then inside the Fusion object you can set what kind of rendering (templatePath) to use, an entryLevel and a maximumLevels properties.

Fusion code:

```
mainMenu = Menu
mainMenu {
    templatePath = 'resource://VendorName.VendorSite/Private/Templates/FusionObjects/
↪MainMenu.html'
    entryLevel = 1
    maximumLevels = 0
}
```

The example above sets first a templatePath for the mainMenu object, then the level to start finding nodes from is set to level 1. It will only take nodes on the current level because of the property maximumLevels is set to 0.

If you want a custom rendering of my menu items then you need to add a template. This template renders a ul list that has a link to a node.

Full HTML code:

```
{namespace neos=Neos\Neos\ViewHelpers}
<ul class="nav">
    <f:for each="{items}" as="item">
        <li class="menu-item">
            <neos:link.node node="{item.node}" />
        </li>
```

(continues on next page)

(continued from previous page)

```
    </f:for>
</ul>
```

What is done is first to include a viewhelper to being able to link my nodes inside the HTML. The namespace in the example is neos to clarify from where the viewhelper is taken.

Viewhelper include:

```
{namespace neos=Neos\Neos\ViewHelpers}
```

The next thing is to iterate through the nodes found by Fusion.

Iterating through nodes:

```
<f:for each="{items}" as="item">
    ...
</f:for>
```

What then is done inside the iteration is that first we wrap our node with a li tag with a class called menu-item. Then we use our viewhelper to (which namespace is neos in this example) link it to a node in Neos. The linking is set in the parameter node, the you can chose what should be shown as a text for the link. In this case the label (default) of the node is the text.

Wrapping and linking of node:

```
<li class="menu-item">
    <neos:link.node node="{item.node}" />
</li>
```

8.12 Rendering a Meta-Navigation

To render a meta navigation (ex: footer navigation) in Neos all you need to use is Fusion and HTML.

A common fact is that most sites have footer where all pages are using the same content or information. So a common issue is how to solve this in the best possible way.

VendorName.VendorSite/Resources/Private/Fusion/Root.fusion

Fusion code:

```
page.body {
    metaMenu = Menu
    metaMenu {
        entryLevel = 2
        templatePath = 'resource://VendorName.VendorSite/Private
        /Templates/FusionObjects/MetaMenu.html'
        maximumLevels = 1
        startingPoint = ${q(site).children('[uriPathSegment="metamenu"]').get(0)}
    }
}
```

The first thing that we define inside the page.body is a Menu object that is called metaMenu. The options available in this example is:

- entryLevel: On which level in the page structure the menu should start.
- templatePath: The path to the template where the rendering is done.

- maximumLevels: How many levels the menu can show.
- startingPoint: The starting point of the menu, in this case the node with name 'nameOfNode' is the starting point.

HTML template code:

```
{namespace neos=Neos\Neos\ViewHelpers}
<nav class="nav">
    <ul class="nav nav-pills">
        <f:for each="{items}" as="item" iteration="menuItemIterator">
            <li class="{item.state}">
                <neos:link.node node="{item.node}" />
            </li>
        </f:for>
    </ul>
</nav>
```

What is done is first to include a view helper to be able to link to nodes inside the HTML. The namespace in the example is neos to clarify from where the viewhelper is taken.

Viewhelper include:

```
{namespace neos=Neos\Neos\ViewHelpers}
```

The next thing is to iterate through the nodes found by Fusion.

Iterating through nodes:

```
<f:for each="{items}" as="item">
    ...
</f:for>
```

What then is done inside the iteration is that first we wrap our node with a li tag with a class called menu-item. Then we use our view helper to (which namespace is neos that is clarified) link it to a node in Neos. The linking is set in the parameter node, the you can choose what should be shown as a text for the link. In this case the label (default) of the node is the text.

Wrapping and linking of node:

```
<li class="{item.state}">
    <neos:link.node node="{item.node}" />
</li>
```

The last thing to do is to include the meta menu to our page layout(s).

Include meta menu:

```
{parts.metaMenu -> f:format.raw()}
```

8.13 Tagging assets automatically

Uploaded assets like images, documents or media files can be assigned to *Tags* and *AssetCollections* manually in the *Media* module. Especially for sites with many assets it is useful to automate this in order to keep files organized.

8.13.1 Asset Collection based on site

Sites can already be assigned to an *AssetCollection* in the *Sites Management* module. If that is the case, any asset uploaded to a node within that site will automatically be added to the corresponding *AssetCollection*. This is especially useful in order to keep files of multi-site installations separated.

For more fine-granular manipulation the `ContentController::assetUploaded` signal can be used to alter assets based on the node they were attached to:

8.13.2 Hooking into the asset creation

In order to hook into the asset creation, a new signal/slot connection has to be established. For this a new *Package.php* (usually in *Packages/Site/The.Site/Classes/*) has to be added:

Example: Package.php

```
<?php
namespace Some\Package;

use Neos\Flow\Core\Bootstrap;
use Neos\Flow\Package\Package as BasePackage;
use Neos\Neos\Controller\Backend\ContentController;

class Package extends BasePackage
{
    public function boot(Bootstrap $bootstrap)
    {
        $dispatcher = $bootstrap->getSignalSlotDispatcher();
        $dispatcher->connect(ContentController::class, 'assetUploaded',␣
↪AssetManipulator::class, 'manipulateAsset');
    }
}
```

Note: If you created a new `Package.php` file you need to run *./flow flow:package:rescan* in order for Flow to pick it up!

The slot gets called with the following arguments:

- The `Asset` instance that is about to be persisted
- The `NodeInterface` instance the asset has been attached to
- The node property name (`string`) the asset has been assigned to

So the signature of the slot method could look like this:

```
function theSlot(Asset $asset, NodeInterface $node, string $propertyName)
```

This allows for manipulation of the asset based on the node property it has been assigned to.

8.13.3 Example: Tagging employee images

Imagine you have a node type *Employee* with the following setup:

```
'Some.Package:Employee':
  superTypes:
    'Neos.Neos:Content': true
  ui:
    label: 'Employee'
    inspector:
      groups:
        'employee':
          label: 'Employee'
  properties:
    'image':
      type: 'Neos\Media\Domain\Model\ImageInterface'
      ui:
        label: 'Employee profile picture'
        reloadIfChanged: true
        inspector:
          group: 'employee'
          editorOptions:
            features:
              mediaBrowser: false
```

The following code would automatically tag this with the *employee* tag (if it exists):

Example: AssetManipulator.php

```
<?php
namespace Some\Package;

use Neos\ContentRepository\Domain\Model\NodeInterface;
use Neos\Flow\Annotations as Flow;
use Neos\Media\Domain\Model\Asset;
use Neos\Media\Domain\Repository\TagRepository;

/**
 * @Flow\Scope("singleton")
 */
class AssetManipulator
{
    /**
     * @Flow\Inject
     * @var TagRepository
     */
    protected $tagRepository;

    public function assignTag(Asset $asset, NodeInterface $node, string $propertyName)
    {
        if (!$node->getNodeType()->isOfType('Some.Package:Employee') || $propertyName␣
↪!== 'image') {
            return;
        }
        $employeeTag = $this->tagRepository->findOneByLabel('employee');
        if ($employeeTag === null) {
            return;
        }
```

(continues on next page)

(continued from previous page)

```
            $asset->addTag($employeeTag);
        }
}
```

Alternatively, the slot could also assign the asset to *AssetCollections* or alter the asset's *title* or *caption*.

8.14 Translating content

Translations for content are based around the concept of *Content Dimensions*. The dimension `language` can be used for most translation scenarios. This cookbook shows how to set up the dimension, migrate existing content to use dimensions and how to work with translations.

8.14.1 Dimension configuration

The first step is to configure a `language` dimension with a *dimension preset* for each language. This should be done in the file `Configuration/Settings.yaml` of your site package:

```
Neos:
  ContentRepository:
    contentDimensions:
      'language':
        label: 'Language'
        icon: 'icon-language'
        default: 'en'
        defaultPreset: 'en'
        presets:
          'en':
            label: 'English'
            values: ['en']
            uriSegment: 'english'
          'fr':
            label: 'Français'
            values: ['fr', 'en']
            uriSegment: 'francais'
          'de':
            label: 'Deutsch'
            values: ['de', 'en']
            uriSegment: 'deutsch'
```

This will configure a dimension `language` with a default dimension value of `en`, a default preset `en` and some presets for the actual available dimension configurations. Each of these presets represents one language that is available for display on the website.

As soon as a dimension with presets is configured, the content module will show a dimension selector to select presets for each dimension. This can be used in combination with a language menu on the website.

8.14.2 Migration of existing content

Existing content of a site needs to be migrated to use the dimension default value, otherwise no nodes would be found. This can be done with a node migration which is included in the `Neos.ContentRepository` package:

```
./flow node:migrate 20150716212459
```

This migration has to be applied whenever a new dimension is configured to set the default value on all existing nodes.

8.14.3 Integrate Language Menu

A simple language menu can be displayed on the site by using the `Neos.Neos:DimensionsMenu` Fusion object:

```
page {
    body {
        parts {
            languageMenu = Neos.Neos:DimensionsMenu {
                dimension = 'language'
            }
        }
    }
}
```

This will render a `<ul>` with links to node variants in other languages of the current document node with a label from a dimension preset. Of course the template can be customized for custom output with the `templatePath` property.

8.14.4 Working with translated content

All content that needs to be translated should go into the default preset first. After selecting a different preset either using the dimension selector or a language menu, the default content will shine through. As soon as a shine-through node is updated, it will be automatically copied to a new node variant with the most specific dimension value in the fallback list.

8.15 Wrapping a List of Content Elements

Create a simple Wrapper that can contain multiple content Elements.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml)

```
'Vendor:Box':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    group: structure
    label: Box
    icon: icon-columns
    inlineEditable: true
  childNodes:
    column0:
      type: 'Neos.Neos:ContentCollection'
```

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/NodeTypes.fusion)

```
prototype(Vendor:Box) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://Vendor.Site/Private/Templates/FusionObjects/Box.
↪html'
        columnContent = Neos.Neos:ContentCollection
        columnContent {
                nodePath = 'column0'
        }
}
```

Html (Sites/Vendor.Site/Private/Templates/FusionObjects/Box.html)

```
{namespace fusion=Neos\Fusion\ViewHelpers}

<div class="container box">
        <div class="column">
                <fusion:render path="columnContent" />
        </div>
</div>
```

8.15.1 Extending it to use an option

You can even simply extend the box to provide a checkbox for different properties.

Yaml (Sites/Vendor.Site/Configuration/NodeTypes.yaml)

```
'Vendor:Box':
  superTypes:
    'Neos.Neos:Content': TRUE
  ui:
    group: structure
    label: Box
    icon: icon-columns
    inlineEditable: TRUE
    inspector:
      groups:
        display:
          label: Display
          position: 5
  properties:
    collapsed:
      type: boolean
      ui:
        label: Collapsed
        reloadIfChanged: TRUE
        inspector:
          group: display
  childNodes:
    column0:
      type: 'Neos.Neos:ContentCollection'
```

Fusion (Sites/Vendor.Site/Resources/Private/Fusion/NodeTypes.fusion)

```
prototype(Vendor:Box) < prototype(Neos.Neos:Content) {
        templatePath = 'resource://Vendor.Site/Private/Templates/FusionObjects/Box.
↪html'
        columnContent = Neos.Neos:ContentCollection
```

(continues on next page)

(continued from previous page)

```
        columnContent {
                nodePath = 'column0'
        }
        collapsed = ${q(node).property('collapsed')}
}
```

Html (Sites/Vendor.Site/Private/Templates/FusionObjects/Box.html)

```
{namespace fusion=Neos\Fusion\ViewHelpers}

<f:if condition="{collapsed}">
        <button>open the collapsed box via js</button>
</f:if>
<div class="container box {f:if(condition: collapsed, then: 'collapsed', else: '')}">
        <div class="column">
                <fusion:render path="columnContent" />
        </div>
</div>
```

CHAPTER

NINE

NEOS OPERATIONS

9.1 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 identified by "username"
  related to the given provider. However, once a user has been found, the new role␣
↪will be added to all
```

(continues on next page)

(continued from previous page)

```
existing accounts related to that user, regardless of its authentication provider.
```

9.1.1 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)

9.1.2 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

9.1.3 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

TEN

APPENDIXES

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex