Neos Media

Release 7.3.x

Neos Team and Contributors

Apr 19, 2024

CONTENTS

1 Variant Presets **3**
- 1.1 Introduction . . . 3
- 1.2 Variants vs. Thumbnails . . . 3
- 1.3 Configuration . . . 4

2 Thumbnail Presets **7**
- 2.1 Introduction . . . 7
- 2.2 Configuration . . . 7
- 2.3 Optimization . . . 7
- 2.4 Utilities . . . 7

3 Asynchronous Thumbnail Generation **9**
- 3.1 Introduction . . . 9
- 3.2 Usage . . . 9
- 3.3 Configuration . . . 9
- 3.4 Optimization . . . 9

4 Thumbnail Generator **11**
- 4.1 Introduction . . . 11
- 4.2 Configuration . . . 11
- 4.3 Build your own Generator . . . 12

5 Configure image generation **15**
- 5.1 Changing output quality of images . . . 15
- 5.2 Convert CMYK images into RGB . . . 15
- 5.3 Changed default filter for Image processing . . . 15
- 5.4 Produce interlaced images . . . 16

6 Asset Privileges **17**
- 6.1 Introduction . . . 17
- 6.2 Complete Example: . . . 18

7 Developer Information **21**
- 7.1 Asset Usage Strategies . . . 21
- 7.2 Extend Asset Validation . . . 22

Neos Media package is a helper package to work with Assets in application based on Flow Framework or Neos CMS.

Neos Media package is licensed under the GPL.

This version of the documentation covering Neos Media 7.3.x has been rendered at: Apr 19, 2024

CHAPTER

ONE

VARIANT PRESETS

1.1 Introduction

Neos Media provides a way to automatically generate variants of an original, based on configuration. This allows for creating images with different aspect ratios, or other adjustments, without further action of an editor.

For example, you may want to generate a square and a wide variant of a given original image in order to use them on your website. Using Fusion, you can access a specific variant generated through a preset, by referring to the preset's identifier and the variant name.

This feature is currently in beta, therefore the API and functionality may change in a future release. It is mainly intended to be combined with further mechanisms, such as automatic image analysis which can set the optimal clipping for an image crop adjustment.

1.2 Variants vs. Thumbnails

The concept of variants and thumbnails are quite similar, but suit different purposes. It's important to know which concept to choose for a specific use case, therefore let's take a quick look at the differences:

An asset, or more specifically, an image, is either imported, uploaded or created. This is what we also call the "original asset". The binary data of this asset is exactly the same which was imported in the first place and it never changes.

An original asset can have any number of variants. Variants are, as the name suggests, variants of an original which can have one or multiple adjustments applied. For example, there may be a variant which only shows a part of the image (by using a "crop adjustment") or one which is a black and white version of the original (using a "grayscale adjustment"). These variants are persistent and can be used like an original asset. The main difference to an original is that

1. they can be modified
2. they are automatically deleted when the original is deleted

Finally, thumbnails are automatically generated previews of an original asset or a variant. They are used to make the representation of an asset more suitable for a specific case, for example by resizing an image to a specific size instead of using the original. Thumbnails are ephemeral, which means that they are created and destroyed automatically, and not manually by an editor.

Based on these concepts, you should use image variant presets, if you want to automate a task which would usually been carried out by an editor.

1.3 Configuration

Variant presets are defined in a Settings.yaml of a given package or distribution. Each preset defines one or more variants to be generated. Each variant can have one or more adjustments automatically applied.

For each preset, one or more media type patterns must be defined. These patterns are regular expressions which are use to match against the concrete IANA media type of a given asset. The configured variants are only created when at least one of the media type patterns matches. Note that you need to specify a complete regular expression, including delimiters ("~" in the example below).

The following example shows the required structure and possible fields of the presets configuration:

```
Neos:
  Media:
    variantPresets:
      'Flownative.Demo:Preset1':
        label: 'Demo Preset 1'
        mediaTypePatterns: ['~image/(jpe?g|png)~', '~image/vnd\.adobe\.photoshop~']
        variants:
          'wide':
            label: 'Wide'
            description: 'An optional description'
            icon: ''
            adjustments:
              crop:
                type: 'Neos\Media\Domain\Model\Adjustment\CropImageAdjustment'
                options:
                  aspectRatio: '16:9'
          'portrait':
            label: 'Portrait'
            description: ''
            icon: ''
            adjustments:
              crop:
                type: 'Neos\Media\Domain\Model\Adjustment\CropImageAdjustment'
                options:
                  aspectRatio: '3:4'
          'square':
            label: 'Square'
            description: ''
            icon: ''
            adjustments:
              crop:
                type: 'Neos\Media\Domain\Model\Adjustment\CropImageAdjustment'
                options:
                  aspectRatio: '1:1'
```

The automatic variant generation for new assets is active by default. It can be disabled via setting:

```
Neos:
  Media:
    autoCreateImageVariantPresets: false
```

To show and edit the variants in the media module the variants tab has to be enabled.

```yaml
Neos:
  Media:
    Browser:
      features:
        variantsTab:
          enable: true
```

CHAPTER

TWO

THUMBNAIL PRESETS

2.1 Introduction

Thumbnail presets allows thumbnails to be easily reused to reduce the amount of rendered thumbnails.

2.2 Configuration

Thumbnails presets are configured using configuration settings in `Neos.Media.thumbnailPresets`. It accepts the parameters used in `ThumbnailConfiguration`, except for the `async` parameter.

```
Neos:
  Media:
    thumbnailPresets:
      'Acme.Demo:Thumbnail':
        maximumWidth: 500
        maximumHeight: 500
```

2.3 Optimization

When new assets are uploaded, thumbnails for the configured presets are automatically created, unless disabled using the configuration setting `Neos.Media.autoCreateThumbnailPresets`.

If Asynchronous Thumbnail Generation is disabled, the thumbnails will be rendered immediately making uploading slower.

2.4 Utilities

To create or clear thumbnails for configured presets use the `typo3.media:media:createthumbnails` and `typo3.media:media:clearthumbnails` commands, see Media Command Reference.

CHAPTER

THREE

ASYNCHRONOUS THUMBNAIL GENERATION

3.1 Introduction

To optimize response times, generation of thumbnails can be done asynchronously. Usage of asynchronous thumbnail generation is determined in the image view helpers usage with the `async` flag. When the flag is used, a link to the thumbnail controller returned instead of rendering the thumbnail if the thumbnail hasn't already been rendered. The thumbnail controller takes a thumbnail object, renders it, if not already done, and redirects to the thumbnail file.

3.2 Usage

To use asynchronous thumbnail generation set the `async` parameter to `TRUE` in the image view helpers, see Media ViewHelper Reference.

3.3 Configuration

The configuration setting `Neos.Media.asyncThumbnails` is used to determine if asynchronous thumbnails are rendered when creating thumbnails for configured Thumbnails Presets.

The setting is additionally used as the default value for the `media:createthumbnails` command, see Media Command Reference.

3.4 Optimization

Since several simultaneous requests for thumbnails can occur, depending on browser and concurrent users, busy servers can experience performance issues. Therefore it is recommended to configure the server to run the command `media:renderthumbnails` often or use a job queue by listening to the `thumbnailCreated` signal and calling `refreshThumbnail` for the thumbnail in the thumbnail service.

Tip: Configure `crontab` to run the render command every minute: `* * * * * /path/to/site/flow media:renderthumbnails`

Use `media:clearthumbnails` and `media:createthumbnails` to refresh thumbnails.

CHAPTER

FOUR

THUMBNAIL GENERATOR

4.1 Introduction

Thumbnail Generators allows previewing different kinds of assets by generating thumbnails for them.

4.1.1 Available Generators

The Neos Media package contains the following generators:

- Document Thumbnail Generator (`DocumentThumbnailGenerator`)

  Generates a Thumbnail for any document type supported by `Imagick`. By default enabled for PDF, EPS and AI (Illustrator).

- Font Thumbnail Generator (`FontThumbnailGenerator`)

  Generates a Thumbnail for any font type supported by `GD`. By default enabled for TTF and ODF.

- Icon Thumbnail Generator (`IconThumbnailGenerator`)

  Returns a static icon image from common types of Assets, based on the asset MIME type.

- Image Thumbnail Generator (`ImageThumbnailGenerator`)

  Generates a Thumbnail for any image types supported by `GD`, `Imagick` or `Gmagick`.

4.2 Configuration

4.2.1 How to configure Generator Priority

In some cases, you need to replace the current Generator by your own implementation or for exemple to replace the PDF Thumbnail Generator by the Icon Generator for a specific project.

You can do that by configuring each Generator priority.

Change the priority of an existing Generator

You can change the priority (higher is better) for an existing Generator, by editing you `Settings.yaml`:

```
Neos:
  Media:
    thumbnailGenerators:
      'Neos\Media\Domain\Model\ThumbnailGenerator\DocumentThumbnailGenerator':
        priority: 100
```

Disabling an existing Generator

To disable an existing Generator use the `disable` configuration option for the desired Generator:

```
Neos:
  Media:
    thumbnailGenerators:
      'Neos\Media\Domain\Model\ThumbnailGenerator\IconThumbnailGenerator':
        disable: true
```

4.2.2 Specific configuration

Check `Settings.yaml` in the Media package to see the available configurations by Generator:

```
'Neos\Media\Domain\Model\ThumbnailGenerator\DocumentThumbnailGenerator':
   resolution: 120
   supportedExtensions: [ 'pdf', 'eps', 'ai' ]
   paginableDocuments: [ 'pdf' ]
```

4.3 Build your own Generator

4.3.1 Implement your own generator

To implement your own Generator, first check the code of the Generators included in the Media package.

Basically, you need to extend `AbstractThumbnailGenerator` and implement the `ThumbnailGeneratorInterface::refresh()` method. The `refresh` method receives a `Thumbnail` object, based on this object do the required processing to generate a thumbnail. In most cases the Thumbnail can be persisted by attaching the new resource to the `Thumbnail` object.

Determine if a Generator can handle the current Thumbnail

You can also implement the `ThumbnailGeneratorInterface::canRefresh()` if your Generator has some specific requirements (like maximum file size, MIME type, external service availability, etc.).

Priority

The `ThumbnailGeneratorStrategy` choose the Generator by two factors, the value of the static property `priority` and the return value of the method `ThumbnailGeneratorInterface::canRefresh()`. For priority value, higher is better:

```
class YourOwnThumbnailGenerator extends AbstractThumbnailGenerator
{
    /**
     * @var integer
     * @api
     */
    protected static $priority = 100;
}
```

You can always override this priority in your `Settings.yaml`.

Configuration

In your generator class use the `AbstractThumbnailGenerator::getOption()` to access your settings:

```
Neos:
  Media:
    thumbnailGenerators:
      'Your\Package\Domain\Model\ThumbnailGenerator\YourOwnThumbnailGenerator':
        priority: 100
        parameterOne: 100
        parameterTwo: 200
```

Remember to add the Media Package in your package `composer.json` to load the Media package before your own:

```
{
    ...
    "require": {
        "neos/flow": "*",
        "neos/media": "*"
    }
    ...
}
```

4.3.2 Community supported Generators

- FilePreviews

  Can be use to integrate the service from filepreviews.io in your project and generate thumbnail for Office or Audio documents.

Feel free to contact us at hello@neos.io, if you publish some Generators under an open-source licence.

CHAPTER

FIVE

CONFIGURE IMAGE GENERATION

5.1 Changing output quality of images

You can change the output quality of generated images within your Settings.yaml. Set the *quality* to your preferred value (between 0 - very poor and 100 - very good).

```
Neos:
  Media:
    image:
      defaultOptions:
        'quality': 90
```

5.2 Convert CMYK images into RGB

If you are working with CMYK images and don't like to convert them automatically into RGB for any reason, you can deactivate this within your Settings.yaml:

```
Neos:
  Media:
    image:
      defaultOptions:
        convertCMYKToRGB: false #default is true
```

5.3 Changed default filter for Image processing

If you have configured a Imagine driver that support alternative filter (this the case is you use Imagick or Gmagick), you can select the filter within your Settings.yaml:

```
Neos:
  Media:
    image:
      defaultOptions:
        resizeFilter: '%\Imagine\Image\ImageInterface::FILTER_UNDEFINED%'
```

Unfortunatly Gd does not support other filter than the default one. Good candidate can be FILTER_BOX or FILTER_CATROOM. You can check the documentation of your image driver for more informations about each filter. Check the ImagineImageImageInterface to know with filter are supported by Imagine.

5.4 Produce interlaced images

To generate progressive images you can configure the driver within your Settings.yaml:

```
  Neos:
    Media:
      image:
        defaultOptions:
          interlace: '%\Imagine\Image\ImageInterface::INTERLACE_LINE%'

Check the \Imagine\Image\ImageInterface to know with mode are supported by Imagine.
```

CHAPTER

SIX

ASSET PRIVILEGES

6.1 Introduction

Asset privileges allows assets to be restricted based on authenticated roles. This package comes with the following privileges:

6.1.1 Restrict read access to *Assets* based on their *media type*

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':
    'Some.Package:ReadAllPDFs':
      matcher: 'hasMediaType("application/pdf")'
```

6.1.2 Restrict read access to *Assets* based on *Tag*

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':
    'Some.Package:ReadConfidentialAssets':
      matcher: 'isTagged("confidential")'
```

6.1.3 Restrict read access to *Assets* based on *Asset Collection*

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':
    'Some.Package:ReadSpecialAssets':
      matcher: 'isInCollection("some-collection")'
```

Of course you can combine the three matchers like:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':
    'Some.Package:ReadConfidentialPdfs':
      matcher: 'hasMediaType("application/pdf") && isTagged("confidential")'
```

6.1.4 Restrict read access to *Tags* based on *Tag label* or *id*

You can match on the *label* of a Tag:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadTagPrivilege':
    'Some.Package:ReadConfidentialTags':
      matcher: 'isLabeled("confidential")'
```

Or on its technical identifier (UUID):

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadTagPrivilege':
    'Some.Package:ReadConfidentialTags':
      matcher: 'hasId("3e8300a6-e5a7-4c3f-aae6-4d7ce35f2caa")'
```

6.1.5 Restrict read access to *Asset Collections* based on *Collection title* or *id*

You can match on the *title* of an Asset Collection:

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetCollectionPrivilege':
    'Some.Package:ReadSpecialAssetCollection':
      matcher: 'isTitled("some-collection")'
```

Or on its technical identifier (UUID):

```
privilegeTargets:
  'Neos\Media\Security\Authorization\Privilege\ReadAssetCollectionPrivilege':
    'Some.Package:ReadSpecialAssetCollection':
      matcher: 'hasId("7c1e8cbc-9205-406d-a384-f8e9440531ad")'
```

6.2 Complete Example:

Given you have three "groups" and corresponding roles *Some.Package:Group1Editor*, *Some.Package:Group2Editor* and *Some.Package:Group3Editor* as well as an administrative role ``Some.Package:Administrator`.

Now, if you have three "Asset Collections" named *group1*, *group2* and *group3* the following `Policy.yaml` would restrict editors to only see collections and assets corresponding to their role:

```
privilegeTargets:

  'Neos\Media\Security\Authorization\Privilege\ReadAssetPrivilege':

    'Some.Package:Group1.ReadAssets':
      matcher: 'isInCollection("group1")'
    'Some.Package:Group2.ReadAssets':
      matcher: 'isInCollection("group2")'
    'Some.Package:Group3.ReadAssets':
      matcher: 'isInCollection("group3")'
```

(continues on next page)

(continued from previous page)

```
  'Neos\Media\Security\Authorization\Privilege\ReadAssetCollectionPrivilege':

    'Some.Package:Group1.ReadCollections':
      matcher: 'isTitled("group1")'
    'Some.Package:Group2.ReadCollections':
      matcher: 'isTitled("group2")'
    'Some.Package:Group3.ReadCollections':
      matcher: 'isTitled("group3")'

roles:

  'Your.Package:Administrator':
    privileges:
      -
        privilegeTarget: 'Some.Package:Group1.ReadAssets'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group1.ReadCollections'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group2.ReadAssets'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group2.ReadCollections'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group3.ReadAssets'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group3.ReadCollections'
        permission: GRANT

  'Your.Package:Group1Editor':
    privileges:
      -
        privilegeTarget: 'Some.Package:Group1.ReadAssets'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group1.ReadCollections'
        permission: GRANT

  'Your.Package:Group2Editor':
    privileges:
      -
        privilegeTarget: 'Some.Package:Group2.ReadAssets'
        permission: GRANT
      -
        privilegeTarget: 'Some.Package:Group2.ReadCollections'
        permission: GRANT

  'Your.Package:Group3Editor':
    privileges:
```

(continues on next page)

(continued from previous page)

```
    -
      privilegeTarget: 'Some.Package:Group3.ReadAssets'
      permission: GRANT
    -
      privilegeTarget: 'Some.Package:Group3.ReadCollections'
      permission: GRANT
```

CHAPTER

SEVEN

DEVELOPER INFORMATION

7.1 Asset Usage Strategies

It is possible to extend the media handling by defining asset usage strategies. Those strategies can tell the media package if an asset is in used, how many times it is used and how it is used.

An asset usage strategy is already implemented for Neos ContentRepository nodes under the sites root, like document and content nodes. For all other usage scenarios, you need to build your own strategy.

To define your own custom usage strategy you have to implement the `Neos\Media\Domain\Strategy\AssetUsageStrategyInterface`. For convenience you can extend the `Neos\Media\Domain\Strategy\AbstractAssetUsageStrategy`.

7.1.1 Example Strategy

```
use TYPO3\Flow\Annotations as Flow;
use Neos\Media\Domain\Strategy\AbstractAssetUsageStrategy;
use TYPO3\Flow\Persistence\PersistenceManagerInterface;

/**
 * @Flow\Scope("singleton")
 */
class MyCustomAssetUsageStrategy extends AbstractAssetUsageStrategy
{
    /**
     * @Flow\Inject
     * @var PersistenceManagerInterface
     */
    protected $persistenceManager;

    /**
     * @var array
     */
    protected $firstlevelCache = [];

    /**
     * Returns an array of usage reference objects.
     *
     * @param AssetInterface $asset
     * @return array<\Neos\Media\Domain\Model\Dto\UsageReference>
```

(continues on next page)

(continued from previous page)

```
     */
    public function getUsageReferences(AssetInterface $asset)
    {
        $assetIdentifier = $this->persistenceManager->getIdentifierByObject($asset);
        if (isset($this->firstlevelCache[$assetIdentifier])) {
            return $this->firstlevelCache[$assetIdentifier];
        }

        // Your code to find asset usage
        foreach ($usages as $usage) {
            $this->firstlevelCache[$assetIdentifier] = new \Neos\Media\Domain\Model\Dto\
↪UsageReference($asset);
        }

        return $this->firstlevelCache[$assetIdentifier];
    }
}
```

7.2 Extend Asset Validation

Imagine you need to extend the validation of assets. For example to prevent duplicate file names or to run copyright checks on images. You can do so by creating your own custom validator. If you make sure that your validator implements the `\Neos\Media\Domain\Validator\AssetValidatorInterface` it will be loaded on object validation. The added errors in your validator will be merged into the model validator of assets.

7.2.1 Example validator

```
<?php
namespace My\Package;

use TYPO3\Flow\Validation\Valwidator\AbstractValidator;
use Neos\Media\Domain\Model\AssetInterface;
use Neos\Media\Domain\Validator\AssetValidatorInterface;

class CustomValidator extends AbstractValidator implements AssetValidatorInterface
{

    /**
     * Check if $value is valid. If it is not valid, needs to add an error
     * to the result.
     *
     * @param AssetInterface $value
     * @return void
     */
    protected function isValid($value)
    {
        // Your object validation
        if ($errors) {
            $this->addError('Some error', 0123456789);
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```