Neos Media Documentation

Release dev-master

The Neos Team

Aug 05, 2017

Contents

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

This version of the documentation covering Neos Media dev-master has been rendered at: Aug 05, 2017

CHAPTER 1

Thumbnail Presets

Introduction

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

Configuration

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

```
TYPO3:
  Media:
    thumbnailPresets:
      'Acme.Demo:Thumbnail':
        maximumWidth: 500
        maximumHeight: 500
```

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.

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 2

Asynchronous Thumbnail Generation

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.

Usage

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

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.

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 3

Thumbnail Generator

Introduction

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

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`.

Configuration

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`:

```
TYPO3:
  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:

```
TYPO3:
  Media:
    thumbnailGenerators:
      'Neos\Media\Domain\Model\ThumbnailGenerator\IconThumbnailGenerator':
        disable: true
```

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' ]
```

Build your own Generator

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:

```
TYPO3:
  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": {
        "typo3/flow": "*",
        "typo3/media": "*"
    }
    ...
}
```

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 4

Configure image generation

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).

```
TYPO3:
  Media:
    image:
      defaultOptions:
        'quality': 90
```

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:

```
TYPO3:
  Media:
    image:
      defaultOptions:
        convertCMYKToRGB: false #default is true
```

CHAPTER 5

Developer Information

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`.

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>
     */
    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];
    }
}
```

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.

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);
        }
    }
}
```