Flask-Imagine Documentation

Release 0.5

Kronas

April 25, 2016

Contents

Extension which provides easy image manipulation support in Flask applications.

This extension allows to alter images in certain ways, such as scaling or rotating them, creating thumbnails, adding watermarks, etc. In order to not hurt application performance, altered images can be cached.

CHAPTER 1

Table of contents

1.1 Getting started

1.1.1 Installation

```
$ pip install Flask-Imagine
```

1.1.2 Configuration example

```
from flask import Flask
from flask.ext.imagine import Imagine

app = Flask(__name__)

app.config['IMAGINE_ADAPTER'] = {
    'name': 'fs',
    'source_folder': 'images',
    'cache_folder': 'cache'
}

app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'cache': True,
        'filters': {
            # Filters initialization parameters
        }
    }
}
```

1.1.3 Dynamic filter sets configuration

```
from flask import Flask
from flask.ext.imagine import Imagine

app = Flask(__name__)

app.config['IMAGINE_ADAPTER'] = {
    'name': 'fs',
```

```
    'source_folder': 'images',
    'cache_folder': 'cache'
}

imagine = Imagine(app)

# Add filter set
imagine.add_filter_set(
    'filter_set_name',
    [
        Filter(parameter='value')  # List of preconfigured filters
    ],
    cached=True
)

# Update existing filter set
imagine.update_filter_set(
    'filter_set_name',
    [
        NewFilter(parameter='value')  # List of preconfigured filters
    ],
    cached=False
)

# Remove existing filter set
imagine.remove_filter_set('filter_set_name')
```

1.2 Configuration guide

```
from flask import Flask
from flask.ext.imagine import Imagine

app = Flask(__name__)

app.config['IMAGINE_URL'] = '/media/cache/resolve'

app.config['IMAGINE_NAME'] = 'imagine'

app.config['IMAGINE_CACHE_ENABLED'] = True

app.config['IMAGINE_ADAPTERS'] = {
    'fs': ImagineFilesystemAdapter
}

app.config['IMAGINE_FILTERS'] = {
    'autorotate': AutorotateFilter,
    'crop': CropFilter,
    'downscale': DownscaleFilter,
    'relative_resize': RelativeResizeFilter,
    'rotate': RotateFilter,
    'thumbnail': ThumbnailFilter,
    'upscale': UpscaleFilter
}

app.config['IMAGINE_ADAPTER'] = {
    'name': 'fs',
```

```
    'source_folder': 'images',
    'cache_folder': 'cache'
}

app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'cache': False,
        'filters': {
            # Filters initialization parameters
        }
    }
}
```

1.3 Storage adapters

1.3.1 Filesystem

Note: Built-in by default

```
app.config['IMAGINE_ADAPTER'] = {
    'name': 'fs',
    'source_folder': 'images',  #  Relative to 'static' folder.
    'cache_folder': 'cache'     #  Optional. Default: 'cache'.
}
```

1.3.2 Amazon AWS S3

Note: Need to install additional package **Flask-Imagine-S3Adapter**

Installation

```
$ pip install Flask-Imagine-S3Adapter
```

Configuration

```
from flask.ext.imagine_s3_adapter import FlaskImagineS3Adapter

app.config['IMAGINE_ADAPTERS'] = {
    's3': FlaskImagineS3Adapter
}

app.config['IMAGINE_ADAPTER'] = {
    'name': 's3',
    'access_key': 'your_access_key',
    'secret_key': 'your_secret_key',
    'bucket_name': 'your_bucket_name',
    'domain': 'domain.tld'      #  Optional. Domain name for using ASW S3 static website hosting.
```

```
    'schema': 'https'           #  Optional.
}
```

1.3.3 Google Cloud Storage

Note: Need to install additional package **Flask-Imagine-GoogleAdapter**

Installation

```
$ pip install Flask-Imagine-GoogleAdapter
```

Configuration

```
from flask.ext.imagine_google_adapter import FlaskImagineGoogleCloudAdapter

app.config['IMAGINE_ADAPTERS'] = {
    'gcs': FlaskImagineGoogleCloudAdapter
}

app.config['IMAGINE_ADAPTER'] = {
    'name': 'gcs',
    'client_id': 'your_client_id',
    'client_secret': 'your_client_secret',
    'bucket_name': 'your_bucket_name',
    'domain': 'domain.tld'      #  Optional. Domain name for using ASW S3 static website hosting.
    'schema': 'https'           #  Optional.
}
```

1.3.4 Microsoft Azure BLOB

Note: Need to install additional package **Flask-Imagine-AzureAdapter**

Installation

```
$ pip install Flask-Imagine-AzureAdapter
```

Configuration

```
from flask.ext.imagine_azure_adapter import FlaskImagineAzureAdapter

app.config['IMAGINE_ADAPTERS'] = {
    'azure': FlaskImagineAzureAdapter
}

app.config['IMAGINE_ADAPTER'] = {
```

```
    'name': 'azure',
    'account_name': 'your_account_name',
    'account_key': 'your_account_key',
    'container_name': 'your_container_name',
    'domain': 'domain.tld'      #  Optional. Domain name for using static website hosting.
    'schema': 'https'           #  Optional.
}
```

1.3.5 Rackspace Files

Note: Need to install additional package **Flask-Imagine-RackspaceAdapter**

Installation

```
$ pip install Flask-Imagine-RackspaceAdapter
```

Configuration

```
from flask.ext.imagine_rackspace_adapter import FlaskImagineRackspaceAdapter

app.config['IMAGINE_ADAPTERS'] = {
    'rackspace': FlaskImagineRackspaceAdapter
}

app.config['IMAGINE_ADAPTER'] = {
    'name': 'rackspace',
    'region': 'your_region',
    'user_name': 'your_user_name',
    'api_key': 'your_api_key',
    'container_name': 'your_container_name'
}
```

1.4 Built-in Filters

1.4.1 Autorotate

It rotates the image automatically to display it as correctly as possible. The rotation to apply is obtained through the metadata stored in the EXIF data of the original image. This filter provides no configuration options:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'autorotate': {}
        }
    }
}
```

1.4.2 Crop

It performs a crop transformation on your image. The start option defines the coordinates of the left-top pixel where the crop begins (the [0, 0] coordinates correspond to the top leftmost pixel of the original image). The size option defines in pixels the width and height (in this order) of the area cropped:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'crop': {'start': [10, 10], 'size': [100, 100]}
        }
    }
}
```

1.4.3 Downscale

It performs a downscale transformation on your image to reduce its size to the given dimensions:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'downscale': {'max': [1920, 1080]}
        }
    }
}
```

A smaller image will be left as it.

1.4.4 Relative resize

The relative_resize filter may be used to heighten, widen, increase or scale an image with respect to its existing dimensions.

```
app.config['IMAGINE_FILTER_SETS'] = {
    'my_heighten': {
        'filters': {
            'relative_resize': {'heighten': 60} #  Transforms 50x40 to 75x60
        }
    },
    'my_widen': {
        'filters': {
            'relative_resize': {'widen': 32}    #  Transforms 50x40 to 32x26
        }
    },
    'my_increase': {
        'filters': {
            'relative_resize': {'increase': 10} #  Transforms 50x40 to 60x50
        }
    },
    'my_decrease': {
        'filters': {
            'relative_resize': {'decrease': 10} #  Transforms 50x40 to 40x30
        }
    },
    'my_scale': {
        'filters': {
```

```
            'relative_resize': {'scale': 2.5}    #  Transforms 50x40 to 125x100
        }
    }
}
```

1.4.5 Rotate

It rotates the image based on specified **angle** (in degrees). The value of the angle configuration option must be a positive integer or float number:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'rotate': {'angle': 90}
        }
    }
}
```

1.4.6 Thumbnail

The thumbnail filter, as the name implies, performs a thumbnail transformation on your image.

The mode can be either outbound or inset. Option inset does a relative resize, where the height and the width will not exceed the values in the configuration. Option outbound does a relative resize, but the image gets cropped if width and height are not the same.

Given an input image sized 50x40 (width x height), consider the following annotated configuration examples:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'thumb_in': {
        'filters': {
            # Transforms 50x40 to 32x26, no cropping
            'thumbnail': {'size': [32, 32], 'mode': 'inset'}
        }
    },
    'thumb_out': {
        'filters': {
            # Transforms 50x40 to 32x32, while cropping the width
            'thumbnail': {'size': [32, 32], 'mode': 'outbound'}
        }
    }
}
```

A smaller image will be left as it. This means you may get images that are smaller than the specified dimensions.

1.4.7 Upscale

It performs an upscale transformation on your image to increase its size to the given dimensions:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'upscale': {'min': [800, 600]}
        }
```

```
    }
}
```

A biggest image will be left as it.

1.4.8 Watermark

The watermark filter pastes a second image onto your image while keeping its ratio. Configuration looks like this:

```
app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'watermark': {
                # Relative to 'static' folder
                'image': 'images/watermark.png',
                # Size of the watermark relative to the origin images size (between 0 and 1)
                'size': 0.5,
                # Position: One of top_left, top, top_right, left, center, right, bottom_left, botto
                'position': 'center',
                # The watermark opacity (between 0 and 1), default: 0.3
                'opacity': 0.3
            }
        }
    }
}
```

Important: Please note that position of watermark filter is important. If you have some filters like *Crop* after it is possible that watermark image will be **cropped**.

1.5 Load your Custom Storage Adapters

The Flask-Imagine allows you to load your own custom storage adapter classes. The only requirement is that each adapter implements the following interface:

```
from flask.ext.imagine.adapters.interface import ImagineAdapterInterface

class MyCustomStorageAdapter(ImagineAdapterInterface):
    configuration_parameter = None

    def __init__(self, configuration_parameter, **kwargs):
        self.configuration_parameter = configuration_parameter

    def get_item(self, path):
        """Get original image"""
        return PIL.Image

    def create_cached_item(self, path, content):
        """Create cached resource item"""
        return str(web_path_to_resource)

    def get_cached_item(self, path):
        """Get cached resource item"""
        return PIL.Image
```

```
    def check_cached_item(self, path):
        """Check for cached resource item exists"""
        return bool()

    def remove_cached_item(self, path):
        """Remove cached resource item"""
        return bool()
```

You can now reference and use your custom storage adapter in your configuration:

```
app.config['IMAGINE_ADAPTERS'] = {
    'my_custom_adapter': MyCustomStorageAdapter
}

app.config['IMAGINE_ADAPTER'] = {
    'name': 'my_custom_adapter',
    'configuration_parameter': 'configuration_parameter_value'
}
```

1.6 Load your Custom Filters

The Flask-Imagine allows you to load your own custom filter classes. The only requirement is that each filter loader implements the following interface:

```
from flask.ext.imagine.filters.interface import ImagineFilterInterface

class MyCustomFilter(ImagineFilterInterface):
    configuration_parameter = None

    def __init__(self, configuration_parameter, **kwargs):
        self.configuration_parameter = configuration_parameter

    def apply(self, resource):
        # Some adjustments

        return resource
```

You can now reference and use your custom filter when defining filter sets you'd like to apply in your configuration:

```
app.config['IMAGINE_FILTERS'] = {
    'my_custom_filter': MyCustomFilter
}

app.config['IMAGINE_FILTER_SETS'] = {
    'filter_set_name': {
        'filters': {
            'my_custom_filter': {'configuration_parameter': 'configuration_parameter_value'}
        }
    }
}
```