MADAM Documentation

Release 0.18+0.gbb6cee7.dirty

Michael Seifert, Erich Seifert

Dec 05, 2019

Contents

CHAPTER 1

Introduction

1.1 Welcome to MADAM

MADAM is a digital asset management library. It aims to facilitate the handling of image, audio, and video files by helping out with several tasks:

Processing MADAM has a very extensible processing architecture. With the various operators offered by MADAM, for example, images can be optimized for display on mobile devices by resizing them, converting them to a different file format, or sharpening them.

Organization MADAM helps you to organize media data by customizable backends to read and store media files. Once the files are stored, the backend can be queried based on the metadata that is present in the media files (e.g. XMP, Exif, . . .) or based on derived properties such as file size or the duration of a sound file.

The *Overview* section will give an introduction to the concepts and vocabulary used in MADAM. If you rather want to get started immediately, have a look at the *Quickstart* section.

1.2 File format support

MADAM supports a wide range of file formats for video, image, and audio data. This is achieved by using several external open source libraries. Pillow and exiv2 (for metadata) are used to read, process, or write image files. FFmpeg is used to read, process, or write audio and video files.

Note: The support of file formats heavily depends on the configuration of your local system. The formats shown in the following list should be available on most configurations. However, it represents only a fraction of the formats supported by the underlying libraries.

Audio

- MP3
- OGG

- WAV / RIFF WAVE

Image

- PNG
- JPEG / JFIF
- GIF

Video

- AVI
- Matroska (MKV), WebM
- MPEG2 transport stream
- OGG
- Quicktime, MPEG4

Vector graphics

- SVG

Adding support for a new format often just means adding a mapping of the library format name to a MIME type to one of the existing processors. If you want to integrate a new library or tool into MADAM, a new `madam.core.Processor` or `madam.core.MetadataProcessor` will have to be implemented. See *Overview* section for more details.

CHAPTER 2

Overview

2.1 Main registry

The class *madam.core.Madam* manages the extensions that can be used to process different file formats. It provides convenience methods to read and to write files.

2.2 Media assets

At the core of MADAM are **assets** in the form of *madam.core.Asset* objects. Asset objects are immutable and provide access the raw data via the file-like attribute `essence` and to the metadata via the dictionary `metadata`.

2.3 Processors

The extensions used to read, process, and write file formats are called **processors**. Usually, they are interfaces to external libraries that are used in the background to do all the heavy lifting. There are two types of processors in MADAM:

Essence processors (or just processors) Represented by *madam.core.Processor* objects. Essence processors are responsible to read and write the actual data in a specific file format. They also offer various operations that can be performed to modify the data, e.g. to resize or to rotate an image. One implementation of this interface is the *madam.image.PillowProcessor* class.

Metadata processors Represented by *madam.core.MetadataProcessor* objects. Metadata processors are responsible to read and write metadata only. Prominent examples of such metadata could be ID3 in MP3 audio files, or Exif in JPEG images. For example, the implementation for Exif metadata is the `madam.exiv2.Exiv2MetadataProcessor` class.

2.4 Operators

Essence processors provide methods to modify assets, which are called **operators**. As operations are usually performed on many media assets, operators are implemented as partial methods that can be pre-configured and then applied to one or many assets.

Note: Operators can raise exceptions of the type *madam.core.OperatorError* if something goes wrong.

2.5 Pipelines

The utility class *madam.core.Pipeline* makes it easy to apply a sequence of operators to one or many assets.

```
portrait_pipeline = Pipeline()
portrait_pipeline.add(processor.resize(width=300, height=300, mode=ResizeMode.FIT))
portrait_pipeline.add(processor.sharpen())

for processed_asset in portrait_pipeline.process(*portrait_assets):
    with open(processed_asset.filename, 'wb') as file:
        file.write(processed_asset.essence.read())
```

2.6 Storage

In MADAM, media assets are organized using modular **storage backends**. Backends have to subclass *madam.core.AssetStorage* and behave like Python dictionaries. It will store a media asset together with its metadata and a set of tags using a unique key. The basic expression to store an asset would be `backend[asset_key] = asset, tags`. Here is a short explanation of the elements in this expression:

- **asset_key** is a unique value. Its data type depends on the storage backend.
- The **asset** is an *madam.core.Asset* object with essence and metadata.
- The set **tags** stores strings that can be used to filter assets.

Storage bakends also support filtering of assets by metadata or tags with the methods *madam.core.AssetStorage.filter()* and *madam.core.AssetStorage.filter_by_tags()*.

Note: Two basic backend implementations are provided:

- *madam.core.InMemoryStorage* uses a Python dictionary to store assets
- *madam.core.ShelveStorage* uses Python `shelve` module to store a serialized version of all assets and tags on disk

---

CHAPTER 3

Quickstart

3.1 Installation

MADAM makes use of other software, which needs to be installed on your system. Make sure you have the following packages installed:

- FFmpeg >=0.9
- libexiv2 >=0.20 (with header files)
- boost.python >=1.48 (with header files)

After you installed these, MADAM can be installed by grabbing the latest release from PyPI:

```
pip install madam
```

3.2 Usage

Initialization:

```
>>> from madam import Madam
>>> manager = Madam()
```

Reading a JPEG image and extracting metadata:

```
>>> with open('path/to/file.jpg', 'rb') as file:
...     asset = manager.read(file)
>>> asset.mime_type
'image/jpeg'
>>> asset.width
800
>>> asset.height
600
```

Changing the size of an image asset:

```
>>> processor = manager.get_processor(asset.essence)
>>> make_thumbnail = processor.resize(width=100, height=100)
>>> resized_asset = make_thumbnail(asset)
>>> resized_asset.width
100
>>> resized_asset.height
100
```

Converting an image to a different file format and saving it to a file:

```
>>> convert_to_png = processor.convert(mime_type='image/png')
>>> png_asset = convert_to_png(asset)
>>> with open('path/to/file.png', 'wb') as file:
...     madam.write(png_asset, file)
```

CHAPTER 4

Module reference

The main module of MADAM is `madam`.It contains several submodules.

4.1 `madam.core` module

class madam.core.**Asset**(*essence*, ***metadata*)
Bases: `object`

Represents a digital asset.

An *Asset* is an immutable value object whose contents consist of *essence* and *metadata*. Essence represents the actual data of a media file, such as the color values of an image, whereas the metadata describes the essence.

Assets should not be instantiated directly. Instead, use *`read()`* to retrieve an *Asset* representing the content.

__init__(*essence*, ***metadata*)
Initializes a new *Asset* with the specified essence and metadata.

Parameters

- **essence** (*file-like object*) – The essence of the asset as a file-like object
- ****metadata** – The metadata describing the essence

essence
Represents the actual content of the asset.

The essence of an MP3 file, for example, is only comprised of the actual audio data, whereas metadata such as ID3 tags are stored separately as metadata.

class madam.core.**AssetStorage**
Bases: `collections.abc.MutableMapping`

Represents an abstract base class for data stores of *`Asset`* objects.

All implementations of *AssetStorage* require a constructor.

The persistence guarantees for stored data may differ based on the respective storage implementation.

__init__()
Initializes a new *AssetStorage*.

clear() → None. Remove all items from D.

filter(***kwargs*)
Returns a sequence of asset keys whose assets match the criteria that are specified by the passed arguments.

Parameters ****kwargs** – Criteria defined as keys and values

Returns Sequence of asset keys

Return type list

filter_by_tags(**tags*)
Returns a set of all asset keys in this storage that have at least the specified tags.

Parameters ***tags** – Mandatory tags of an asset to be included in result

Returns Keys of the assets whose tags are a superset of the specified tags

Return type set

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class madam.core.**InMemoryStorage**
Bases: *madam.core.AssetStorage*

Represents a non-persistent storage backend for *Asset* objects.

Assets are not serialized, but stored in memory.

__init__()
Initializes a new, empty *InMemoryStorage* object.

clear() → None. Remove all items from D.

filter(***kwargs*)
Returns a sequence of asset keys whose assets match the criteria that are specified by the passed arguments.

Parameters ****kwargs** – Criteria defined as keys and values

Returns Sequence of asset keys

Return type list

filter_by_tags(**tags*)
Returns a set of all asset keys in this storage that have at least the specified tags.

Parameters ***tags** – Mandatory tags of an asset to be included in result

Returns Keys of the assets whose tags are a superset of the specified tags

Return type set

get (*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items () → a set-like object providing a view on D's items

keys () → a set-like object providing a view on D's keys

pop (*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem () → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault (*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update ([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values () → an object providing a view on D's values

class madam.core.**Madam**
Bases: object

Represents an instance of the library.

__init__ ()
Initializes a new library instance with default configuration.

The default configuration includes a list of all available Processor and MetadataProcessor implementations.

get_processor (*file*)
Returns a processor that can read the data in the specified file.

Parameters **file** (*file-like object*) – file-like object to be parsed.

Returns Processor object that can handle the data in the specified file, or None if no suitable processor could be found.

Return type *Processor*

read (*file*, *additional_metadata=None*)
Reads the specified file and returns its contents as an *Asset* object.

Parameters

- **file** (*file-like object*) – file-like object to be parsed
- **additional_metadata** (*dict*) – optional metadata for the resulting asset. Existing metadata entries extracted from the file will be overwritten.

Returns Asset representing the specified file

Return type *Asset*

Raises

- ***UnsupportedFormatError*** – if the file format cannot be recognized or is not supported
- **TypeError** – if the file is None

Example

```
>>> import io
>>> from madam import Madam
>>> manager = Madam()
>>> file = io.BytesIO(b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00\x01\x00\x00\x00\x01'
... b
↪'\x08\x06\x00\x00\x00\x1f\x15\xc4\x89\x00\x00\x00\nIDATx\x9cc\x00\x01\x00\x00\x05\x00
↪'
... b'\x01\r\n-\xb4\x00\x00\x00\x00IEND\xaeB`\x82')
>>> asset = manager.read(file)
```

write(*asset*, *file*)
Write the *Asset* object to the specified file.

Parameters

- **asset** (*Asset*) – Asset that contains the data to be written
- **file** (*file-like object*) – file-like object to be written

Example

```
>>> import io
>>> import os
>>> from madam import Madam
>>> from madam.core import Asset
>>> gif_asset = Asset(essence=io.BytesIO(b'GIF89a\x01\x00\x01\x00\x00\x00\x00;
↪'), mime_type='image/gif')
>>> manager = Madam()
>>> with open(os.devnull, 'wb') as file:
...     manager.write(gif_asset, file)
>>> wav_asset = Asset(
...     essence=io.BytesIO(b'RIFF$\x00\x00\x00WAVEfmt␣
↪\x10\x00\x00\x00\x01\x00\x01\x00D\xac'
...                b'\x00\x00\x88X\x01\x00\x02\x00\x10\x00data\x00\x00\x00\x00'),
...     mime_type='video/mp4')
>>> with open(os.devnull, 'wb') as file:
...     manager.write(wav_asset, file)
```

class madam.core.**MetadataProcessor**
Bases: object

Represents an entity that can manipulate metadata.

Every *MetadataProcessor* needs to have a no-args *__init__* method in order to be registered correctly.

__init__()
Initializes a new *MetadataProcessor*.

combine(*file*, *metadata*)
Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

- **metadata** (*dict*) – Mapping of the metadata format to the metadata dict
- **file** (*file-like object*) – Container file

Returns file-like object with combined content

Return type io.BytesIO

formats
The metadata formats which are supported.

Returns supported metadata formats

Return type set[str]

read(*file*)
Reads the file and returns the metadata.

The metadata that is returned is grouped by type. The keys are specified by `format`.

Parameters **file** (*file-like object*) – File-like object to be read

Returns Metadata contained in the file

Return type dict

Raises ***UnsupportedFormatError*** – if the data is corrupt or its format is not supported

strip(*file*)
Removes all metadata of the supported type from the specified file.

Parameters **file** (*file-like object*) – file-like that should get stripped of the metadata

Returns file-like object without metadata

Return type io.BytesIO

exception madam.core.**OperatorError**(**args*, ***kwargs*)
Bases: Exception

Represents an error that is raised whenever an error occurs in an *operator()*.

__init__(**args*, ***kwargs*)
Initializes a new *OperatorError*.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class madam.core.**Pipeline**
Bases: object

Represents a processing pipeline for *Asset* objects.

The pipeline can be configured to hold a list of asset processing operators, all of which are applied to one or more assets when calling the *process()* method.

__init__()
Initializes a new pipeline without operators.

add(*operator*)
Appends the specified operator to the processing chain.

Parameters **operator** – Operator to be added

process(**assets*)
Applies the operators in this pipeline on the specified assets.

Parameters ***assets** (Asset) – Asset objects to be processed

Returns Generator with processed assets

class madam.core.**Processor**
Bases: object

Represents an entity that can create *Asset* objects from binary data.

Every *Processor* needs to have a no-args *__init__* method in order to be registered correctly.

__init__()
Initializes a new *Processor*.

can_read(*file*)
Returns whether the specified MIME type is supported by this processor.

Parameters **file** (*file-like object*) – file-like object to be tested

Returns whether the data format of the specified file is supported or not

Return type bool

read(*file*)
Returns an *Asset* object whose essence is identical to the contents of the specified file.

Parameters **file** (*file-like object*) – file-like object to be read

Returns Asset with essence

Return type *Asset*

Raises ***UnsupportedFormatError*** – if the specified data format is not supported

class madam.core.**ShelveStorage**(*path*)
Bases: *madam.core.AssetStorage*

Represents a persistent storage backend for *Asset* objects. Asset keys must be strings.

ShelveStorage uses a file on the file system to serialize Assets.

__init__(*path*)
Initializes a new *ShelveStorage* with the specified path.

Parameters **path** (*pathlib.Path or str*) – File system path where the data should be stored

clear() → None. Remove all items from D.

filter(***kwargs*)
Returns a sequence of asset keys whose assets match the criteria that are specified by the passed arguments.

Parameters ****kwargs** – Criteria defined as keys and values

Returns Sequence of asset keys

Return type list

filter_by_tags(**tags*)
Returns a set of all asset keys in this storage that have at least the specified tags.

Parameters ***tags** – Mandatory tags of an asset to be included in result

Returns Keys of the assets whose tags are a superset of the specified tags

Return type set

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

exception madam.core.**UnsupportedFormatError**(**args*, ***kwargs*)
Bases: Exception

Represents an error that is raised whenever file content with unknown type is encountered.

__init__(**args*, ***kwargs*)
Initializes a new *UnsupportedFormatError*.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

madam.core.**operator**(*function*)
Decorator function for methods that process assets.

Usually, it will be used with operations in a *Processor* implementation to make the methods configurable before applying the method to an asset.

Only keyword arguments are allowed for configuration.

Example for using a decorated `convert` method:

```
convert_to_opus = processor.convert(mime_type='audio/opus')
convert_to_opus(asset)
```

Parameters **function** – Method to decorate

Returns Configurable method

4.2 madam.exiv2 module

4.3 madam.ffmpeg module

class madam.ffmpeg.**FFmpegMetadataProcessor**
Bases: *madam.core.MetadataProcessor*

Represents a metadata processor that uses FFmpeg.

__init__()
Initializes a new *FFmpegMetadataProcessor*.

combine(*file*, *metadata_by_type*)
Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

- **metadata** (*dict*) – Mapping of the metadata format to the metadata dict
- **file** (*file-like object*) – Container file

Returns file-like object with combined content

Return type io.BytesIO

formats
The metadata formats which are supported.

Returns supported metadata formats

Return type set[str]

read(*file*)
Reads the file and returns the metadata.

The metadata that is returned is grouped by type. The keys are specified by `format`.

Parameters **file** (*file-like object*) – File-like object to be read

Returns Metadata contained in the file

Return type dict

Raises ***UnsupportedFormatError*** – if the data is corrupt or its format is not supported

strip(*file*)
Removes all metadata of the supported type from the specified file.

Parameters **file** (*file-like object*) – file-like that should get stripped of the metadata

Returns file-like object without metadata

Return type io.BytesIO

class madam.ffmpeg.**FFmpegProcessor**
Bases: *madam.core.Processor*

Represents a processor that uses FFmpeg to read audio and video data.

The minimum version of FFmpeg required is v0.9.

__init__()
Initializes a new *FFmpegProcessor*.

Raises **EnvironmentError** – if the installed version of ffprobe does not match the minimum version requirement

can_read(*file*)
Returns whether the specified MIME type is supported by this processor.

Parameters **file** (*file-like object*) – file-like object to be tested

Returns whether the data format of the specified file is supported or not

Return type bool

convert(*asset*, *mime_type*, *video=None*, *audio=None*, *subtitle=None*)
Creates a new asset of the specified MIME type from the essence of the specified asset.

Additional options can be specified for video, audio, and subtitle streams. Options are passed as dictionary instances and can contain various keys for each stream type.

Options for video streams:

- **codec** – Processor-specific name of the video codec as string
- **bitrate** – Target bitrate in kBit/s as float number

Options for audio streams:

- **codec** – Processor-specific name of the audio codec as string

- **bitrate** – Target bitrate in kBit/s as float number

Options for subtitle streams:

- **codec** – Processor-specific name of the subtitle format as string

Parameters

- **asset** (*Asset*) – Asset whose contents will be converted
- **mime_type** (*MimeType or str*) – MIME type of the video container
- **video** (*dict or None*) – Dictionary with options for video streams.
- **audio** (*dict or None*) – Dictionary with options for audio streams.
- **subtitle** (*dict or None*) – Dictionary with the options for subtitle streams.

Returns New asset with converted essence

Return type *Asset*

crop(*asset*, *x*, *y*, *width*, *height*)

Creates a cropped video asset whose essence is cropped to the specified rectangular area.

Parameters

- **asset** (*Asset*) – Video asset whose contents will be cropped
- **x** (*int*) – Horizontal offset of the cropping area from left
- **y** (*int*) – Vertical offset of the cropping area from top
- **width** (*int*) – Width of the cropping area
- **height** (*int*) – Height of the cropping area

Returns New asset with cropped essence

Return type *Asset*

extract_frame(*asset*, *mime_type*, *seconds=0*)

Creates a new image asset of the specified MIME type from the essence of the specified video asset.

Parameters

- **asset** (*Asset*) – Video asset which will serve as the source for the frame
- **mime_type** (*MimeType or str*) – MIME type of the destination image
- **seconds** (*float*) – Offset of the frame in seconds

Returns New image asset with converted essence

Return type *Asset*

read(*file*)

Returns an *Asset* object whose essence is identical to the contents of the specified file.

Parameters **file** (*file-like object*) – file-like object to be read

Returns Asset with essence

Return type *Asset*

Raises ***UnsupportedFormatError*** – if the specified data format is not supported

resize(*asset*, *width*, *height*)
Creates a new image or video asset of the specified width and height from the essence of the specified image or video asset.

Width and height must be positive numbers.

Parameters

- **asset** (Asset) – Video asset that will serve as the source for the frame
- **width** (*int*) – Width of the resized asset
- **height** (*int*) – Height of the resized asset

Returns New asset with specified width and height

Return type *Asset*

rotate(*asset*, *angle*, *expand=False*)
Creates an asset whose essence is rotated by the specified angle in degrees.

Parameters

- **asset** (Asset) – Asset whose contents will be rotated
- **angle** (*float*) – Angle in degrees, counter clockwise
- **expand** (*bool*) – If true, changes the dimensions of the new asset so it can hold the entire rotated essence, otherwise the dimensions of the original asset will be used.

Returns New asset with rotated essence

Return type *Asset*

trim(*asset*, *from_seconds=0*, *to_seconds=0*)
Creates a trimmed audio or video asset that only contains the data between from_seconds and to_seconds.

Parameters

- **asset** (Asset) – Audio or video asset, which will serve as the source
- **from_seconds** (*float*) – Start time of the clip in seconds
- **to_seconds** (*float*) – End time of the clip in seconds

Returns New asset with trimmed essence

Return type *Asset*

4.4 madam.image module

class madam.image.**FlipOrientation**
Bases: enum.Enum

Represents an axis for image flip operations.

HORIZONTAL = 0
Horizontal axis

VERTICAL = 1
Vertical axis

class madam.image.**PillowProcessor**

Bases: *madam.core.Processor*

Represents a processor that uses Pillow as a backend.

__init__()

Initializes a new *PillowProcessor*.

auto_orient(*asset*)

Creates a new asset whose essence is rotated according to the Exif orientation. If no orientation metadata exists or asset is not rotated, an identical asset object is returned.

Parameters **asset** (Asset) – Asset with orientation metadata

Returns Asset with rotated essence

Return type *Asset*

can_read(*file*)

Returns whether the specified MIME type is supported by this processor.

Parameters **file** (*file-like object*) – file-like object to be tested

Returns whether the data format of the specified file is supported or not

Return type bool

convert(*asset*, *mime_type*, *color_space=None*, *depth=None*, *data_type=None*)

Creates a new asset of the specified MIME type from the essence of the specified asset.

Parameters

- **asset** (Asset) – Asset whose contents will be converted
- **mime_type** (*MimeType or str*) – Target MIME type
- **color_space** (*str or None*) – Name of color space
- **depth** (*int or None*) – Bit depth per channel
- **data_type** (*str or None*) – Data type of the pixels, e.g. 'uint' or 'float'

Returns New asset with converted essence

Return type *Asset*

crop(*asset*, *x*, *y*, *width*, *height*)

Creates a new asset whose essence is cropped to the specified rectangular area.

Parameters

- **asset** (Asset) – Asset whose contents will be cropped
- **x** (*int*) – horizontal offset of the cropping area from left
- **y** (*int*) – vertical offset of the cropping area from top
- **width** (*int*) – width of the cropping area
- **height** (*int*) – height of the cropping area

Returns New asset with cropped essence

Return type *Asset*

flip(*asset*, *orientation*)

Creates a new asset whose essence is flipped according the specified orientation.

Parameters

- **asset** (Asset) – Asset whose essence is to be flipped
- **orientation** (FlipOrientation) – axis of the flip operation

Returns Asset with flipped essence

Return type *Asset*

read(*file*)

Returns an Asset object whose essence is identical to the contents of the specified file.

Parameters **file** (*file-like object*) – file-like object to be read

Returns Asset with essence

Return type *Asset*

Raises ***UnsupportedFormatError*** – if the specified data format is not supported

resize(*asset*, *width*, *height*, *mode=<ResizeMode.EXACT: 0>*)

Creates a new Asset whose essence is resized according to the specified parameters.

Parameters

- **asset** (Asset) – Asset to be resized
- **width** (*int*) – target width
- **height** (*int*) – target height
- **mode** (ResizeMode) – resize behavior

Returns Asset with resized essence

Return type *Asset*

rotate(*asset*, *angle*, *expand=False*)

Creates an asset whose essence is rotated by the specified angle in degrees.

Parameters

- **asset** (Asset) – Asset whose contents will be rotated
- **angle** (*float*) – Angle in degrees, counter clockwise
- **expand** (*bool*) – If true, changes the dimensions of the new asset so it can hold the entire rotated essence, otherwise the dimensions of the original asset will be used.

Returns New asset with rotated essence

Return type *Asset*

transpose(*asset*)

Creates a new image asset whose essence is the transpose of the specified asset's essence.

Parameters **asset** (Asset) – Image asset whose essence is to be transposed

Returns New image asset with transposed essence

Return type *Asset*

class madam.image.**ResizeMode**

Bases: enum.Enum

Represents a behavior for image resize operations.

EXACT = 0

Image exactly matches the specified dimensions

FILL = 2
Image is resized to completely fill the specified dimensions

FIT = 1
Image is resized to fit completely into the specified dimensions

4.5 madam.mime module

class madam.mime.**MimeType**(*mediatype*, *subtype=None*)
Bases: object

Represents a MIME type according to RFC 2045. This class behaves identical to its string representation in `dict` and `set`.

Limitations:

- Suffixes are considered a part of the subtype
- Parameters like `charset` are not supported and will be treated as part of the subtype

__init__(*mediatype*, *subtype=None*)
Initializes a new MIME type with either

- both, *media type* and *subtype* strings
- a complete MIME type string like `'audio/opus'`
- another `MimeType` object

Parameters

- **mediatype** (*str or MimeType or None*) – Can define either the media type, or the complete MIME type by passing a MIME type string or another `MimeType` instance.
- **subtype** (*str or None*) – Defines the subtype.

4.6 madam.vector module

class madam.vector.**SVGMetadataProcessor**
Bases: *madam.core.MetadataProcessor*

Represents a metadata processor that handles Scalable Vector Graphics (SVG) data.

It is assumed that the SVG XML uses UTF-8 encoding.

__init__()
Initializes a new *SVGMetadataProcessor*.

combine(*file*, *metadata*)
Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

- **metadata** (*dict*) – Mapping of the metadata format to the metadata dict
- **file** (*file-like object*) – Container file

Returns file-like object with combined content

Return type io.BytesIO

formats
The metadata formats which are supported.

Returns supported metadata formats

Return type set[str]

read(*file*)
Reads the file and returns the metadata.

The metadata that is returned is grouped by type. The keys are specified by `format`.

Parameters **file** (*file-like object*) – File-like object to be read

Returns Metadata contained in the file

Return type dict

Raises ***UnsupportedFormatError*** – if the data is corrupt or its format is not supported

strip(*file*)
Removes all metadata of the supported type from the specified file.

Parameters **file** (*file-like object*) – file-like that should get stripped of the metadata

Returns file-like object without metadata

Return type io.BytesIO

class madam.vector.**SVGProcessor**
Bases: *madam.core.Processor*

Represents a processor that handles *Scalable Vector Graphics* (SVG) data.

__init__()
Initializes a new *SVGProcessor*.

can_read(*file*)
Returns whether the specified MIME type is supported by this processor.

Parameters **file** (*file-like object*) – file-like object to be tested

Returns whether the data format of the specified file is supported or not

Return type bool

read(*file*)
Returns an `Asset` object whose essence is identical to the contents of the specified file.

Parameters **file** (*file-like object*) – file-like object to be read

Returns Asset with essence

Return type *Asset*

Raises ***UnsupportedFormatError*** – if the specified data format is not supported

shrink(*asset*)
Shrinks the size of an SVG asset.

Parameters **asset** (`Asset`) – Media asset to be shrunk

Returns Shrunk vector asset

Return type *Asset*

CHAPTER 5

Index

Python Module Index

m

Index

Symbols

A

C

E

F

G

H

I

K

M

O

P

R

S

T

U

V

W