
MADAM Documentation

Release 0.21.0+0.g2226ef7.dirty

Erich Seifert, Michael Seifert

Apr 11, 2021

Contents

1	Introduction	1
1.1	Welcome to MADAM	1
1.2	File format support	1
2	Overview	3
2.1	Main registry	3
2.2	Media assets	3
2.3	Processors	3
2.4	Operators	4
2.5	Pipelines	4
2.6	Storage	4
3	Quickstart	5
3.1	Installation	5
3.2	Usage	5
4	Configuration	7
4.1	Setting default format options	7
4.2	Image options	7
4.3	Video options	8
5	Module reference	11
5.1	madam.core module	11
5.2	madam.exif module	17
5.3	madam.ffmpeg module	18
5.4	madam.image module	21
5.5	madam.mime module	24
5.6	madam.vector module	24
6	Index	27
	Python Module Index	29
	Index	31

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 *piexif* (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.

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.exif.ExifMetadataProcessor` 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
-

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 >=3.3 for video processing

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()
```

Define settings for different file formats:

```
>>> config = {
...     'image/jpeg': dict(
...         quality=85,
...     ),
... }
>>> manager = Madam(config)
```

Reading a JPEG image and extracting metadata:

```
>>> with open('path/to/file.jpg', 'rb') as file:
...     asset = manager.read(file)
```

(continues on next page)

(continued from previous page)

```
>>> 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:
...     manager.write(png_asset, file)
```

4.1 Setting default format options

MADAM has several configuration options that can modify the default settings which are used to write file formats. The configuration is stored in a dictionary that can be passed to `madam.core.Madam.__init__()`. The structure of the dict depends on the file type category (e.g. audio, image, or video).

4.2 Image options

Image settings can be stored for all image formats or for a certain MIME type.

The following example shows how to set the quality factor for all images and how to enable Zopfli compression for PNG images:

```
>>> config = {  
...     'image': {  
...         'quality': 80,  
...     },  
...     'image/png': {  
...         'zopfli': True,  
...     },  
... }
```

The following list shows all available options for file formats.

4.2.1 JPEG (image/jpeg)

progressive A boolean value that defines whether progressive JPEG images should be written.

Defaults to True.

quality A compression factor as an integer in the range between 0 and 100. Smaller factors produce smaller files with lower quality.

Defaults to 80.

4.2.2 PNG (image/png)

zopfli Boolean that defines whether Zopfli should be used to further compress PNG images.

Defaults to True.

Warning: Enabling Zopfli can increase processing times dramatically depending on the input data.

zopfli_strategies String that lists all filter strategies to try when saving PNG images:

- 0–4: Apply PNG filter type 0 to 4 to scanlines
- m: Minimum sum
- e: Entropy
- p: Predefined
- b: Brute force

Defaults to '0me'.

4.2.3 WebP (image/webp)

method An integer in the range between 0 and 6 that defines the quality/speed trade-off when optimizing WebP images. 0 means faster compression but larger files, 6 means slower compression but smaller files.

Defaults to 6.

quality A compression factor as an integer in the range between 0 and 100. Smaller factors produce smaller files with lower quality.

Defaults to 80.

4.3 Video options

Video content is split in two categories: Container options are stored by MIME type and codec options are stored in a separate MIME type category `codec`.

The following example shows how to set the Constant Rate Factor (CRF) for the h.264 codec (using `libx265`):

```
>>> config = {
...     'video/quicktime': {
...         'faststart': True,
...     },
...     'codec/libx265': {
...         'crf': 24,
...     },
... }
```

The following list shows all available options for video containers.

4.3.1 Quicktime/MPEG4 (video/quicktime)

faststart Boolean that defines whether the video and audio files should be written in a way that allows a fast start when streaming.

Defaults to True.

The following list shows all available options for video codecs.

4.3.2 AVC/h.264 (libx264)

crf An integer that defines the Constant Rate Factor (CRF) for quality and rate control in videos. 0 would encode slowly to lossless quality, while 51 would encode fast to the worst quality. A sane range for AVC/h.264 is between 18 and 28.

Defaults to 23.

4.3.3 HEVC/h.265 (libx265)

crf An integer that defines the Constant Rate Factor (CRF) for quality and rate control in videos. 0 would encode slowly to lossless quality, while 51 would encode fast to the worst quality. A sane range for HEVC/h.265 is between 18 and 28.

Defaults to 28.

4.3.4 VP8 (libvpx)

crf An integer that defines the Constant Rate Factor (CRF) for quality and rate control in videos. 0 would encode very slowly to lossless quality, while 63 would encode very fast to the worst quality. A sane range for VP8 is between 4 and 63.

Defaults to 10.

4.3.5 VP9 (libvpx-vp9 or vp9)

crf An integer that defines the Constant Rate Factor (CRF) for quality and rate control in videos. 0 would encode very slowly to lossless quality, while 63 would encode very fast to the worst quality. A sane range for VP9 is between 4 and 63.

Defaults to 32.

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

5.1 `madam.core` module

class `madam.core.Asset` (*essence: IO, **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: IO, **metadata*) → None

Initializes a new *Asset* with the specified essence and metadata.

Parameters

- **essence** (*IO*) – 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`, `typing.Generic`

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__()` → None

Initializes a new *AssetStorage*.

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

`filter(**kwargs)` → Iterable[AssetKey]

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 Iterable

`filter_by_tags(*tags)` → Iterable[AssetKey]

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 Iterable

`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__()` → None

Initializes a new, empty *InMemoryStorage* object.

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

`filter(**kwargs)` → Iterable[AssetKey]

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 Iterable

`filter_by_tags(*tags)` → Iterable[AssetKey]

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 Iterable

get ($k[, d]$) $\rightarrow D[k]$ if k in D , else d . d defaults to `None`.

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

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

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

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

setdefault ($k[, d]$) $\rightarrow D.get(k, d)$, also set $D[k]=d$ if k not in D

update ($[E], **F$) \rightarrow `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 () \rightarrow an object providing a view on D 's values

class `madam.core.Madam` (*config: Optional[Mapping[str, Any]] = None*)

Bases: `object`

Represents an instance of the library.

__init__ (*config: Optional[Mapping[str, Any]] = None*) \rightarrow `None`

Initializes a new library instance with default configuration.

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

Parameters **config** – Mapping with settings.

get_processor (*file: IO*) \rightarrow Optional[`madam.core.Processor`]

Returns a processor that can read the data in the specified file. If no suitable processor can be found `None` will be returned.

Parameters **file** (*IO*) – 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` or `None`

read (*file: IO, additional_metadata: Mapping[KT, VT_co] = None*)

Reads the specified file and returns its contents as an `Asset` object.

Parameters

- **file** (*IO*) – file-like object to be parsed
- **additional_metadata** (*Mapping*) – 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: madam.core.Asset, file: IO*) → None

Write the *Asset* object to the specified file.

Parameters

- **asset** (*Asset*) – Asset that contains the data to be written
- **file** (*IO*) – 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** (*config: Optional[Mapping[str, Any]] = None*)

Bases: *object*

Represents an entity that can manipulate metadata.

Every *MetadataProcessor* needs to have an `__init__` method with an optional *config* parameter in order to be registered correctly.

__init__ (*config: Optional[Mapping[str, Any]] = None*) → None

Initializes a new *MetadataProcessor*.

combine (*file: IO, metadata: Mapping[KT, VT_co]*) → IO

Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

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

Returns file-like object with combined content

Return type IO

formats

The metadata formats which are supported.

Returns supported metadata formats

Return type `set[str]`

read (*file: IO*) → Mapping[KT, VT_co]

Reads the file and returns the metadata.

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

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

Returns Metadata contained in the file

Return type Mapping

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

strip (*file: IO*) → IO

Removes all metadata of the supported type from the specified file.

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

Returns file-like object without metadata

Return type IO

exception `madam.core.OperatorError` (*args)

Bases: `Exception`

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

`__init__` (*args)

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__` () → None

Initializes a new pipeline without operators.

add (*operator: Callable*) → None

Appends the specified operator to the processing chain.

Parameters **operator** – Operator to be added

process (*assets) → Generator[madam.core.Asset, float, None]

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 (config: Optional[Mapping[str, Any]] = None)
```

Bases: `object`

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

Every `Processor` needs to have an `__init__` method with an optional `config` parameter in order to be registered correctly.

```
__init__ (config: Optional[Mapping[str, Any]] = None) → None
```

Initializes a new `Processor`.

Parameters `config` – Mapping with settings.

```
can_read (file: IO) → bool
```

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

Parameters `file` (`IO`) – file-like object to be tested

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

Return type `bool`

```
read (file: IO) → madam.core.Asset
```

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

Parameters `file` (`IO`) – 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: Union[pathlib.Path, str])
```

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: Union[pathlib.Path, str])
```

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) → Iterable[AssetKey]
```

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 `Iterable`

```
filter_by_tags (*tags) → Iterable[AssetKey]
```

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 `Iterable`

```
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*)

Bases: `Exception`

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

__init__ (**args*) → *None*
Initializes a new `UnsupportedFormatError`.

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

`madam.core.operator` (*function*: `Callable[[...], madam.core.Asset]`) → `Callable[[...], Callable[[...], madam.core.Asset]]`

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

5.2 madam.exif module

class `madam.exif.ExifMetadataProcessor` (*config*: `Optional[Mapping[str, Any]] = None`)

Bases: `madam.core.MetadataProcessor`

Represents a metadata processor for Exif metadata.

__init__ (*config*: `Optional[Mapping[str, Any]] = None`) → *None*
Initializes a new `ExifMetadataProcessor`.

Parameters `config` – Mapping with settings

combine (*essence*: *IO*, *metadata_by_format*: `Mapping[str, Mapping[KT, VT_co]]`) → *IO*
Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

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

Returns file-like object with combined content

Return type *IO*

formats

The metadata formats which are supported.

Returns supported metadata formats

Return type *set[str]*

read (*file: IO*) → *Mapping[str, Mapping[KT, VT_co]]*

Reads the file and returns the metadata.

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

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

Returns Metadata contained in the file

Return type *Mapping*

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

strip (*file: IO*) → *IO*

Removes all metadata of the supported type from the specified file.

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

Returns file-like object without metadata

Return type *IO*

5.3 madam.ffmpeg module

class *madam.ffmpeg.FFmpegMetadataProcessor* (*config: Optional[Mapping[str, Any]] = None*)

Bases: *madam.core.MetadataProcessor*

Represents a metadata processor that uses FFmpeg.

__init__ (*config: Optional[Mapping[str, Any]] = None*) → *None*

Initializes a new *FFmpegMetadataProcessor*.

Parameters **config** – Mapping with settings.

combine (*file: IO, metadata_by_type: Mapping[str, Mapping[KT, VT_co]]*) → *IO*

Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

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

Returns file-like object with combined content

Return type *IO*

formats

The metadata formats which are supported.

Returns supported metadata formats

Return type `set[str]`

read (*file*: *IO*) → `Mapping[str, Mapping[KT, VT_co]]`

Reads the file and returns the metadata.

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

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

Returns Metadata contained in the file

Return type `Mapping`

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

strip (*file*: *IO*) → *IO*

Removes all metadata of the supported type from the specified file.

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

Returns file-like object without metadata

Return type *IO*

class `madam.ffmpeg.FFmpegProcessor` (*config*: `Optional[Mapping[str, Any]] = None`)

Bases: `madam.core.Processor`

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

The minimum version of FFmpeg required is v3.3.

__init__ (*config*: `Optional[Mapping[str, Any]] = None`) → `None`

Initializes a new `FFmpegProcessor`.

Parameters **config** – Mapping with settings.

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

can_read (*file*: *IO*) → `bool`

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

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

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

Return type `bool`

convert (*asset*: `madam.core.Asset`, *mime_type*: `Union[madam.mime.MimeType, str]`, *video*: `Optional[Mapping[str, Any]] = None`, *audio*: `Optional[Mapping[str, Any]] = None`, *subtitle*: `Optional[Mapping[str, Any]] = None`) → `madam.core.Asset`

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: madam.core.Asset, x: int, y: int, width: int, height: int*) → *madam.core.Asset*
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: madam.core.Asset, mime_type: Union[*madam.mime.MimeType*, *str*], seconds: float = 0*) → *madam.core.Asset*
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: IO*) → *madam.core.Asset*

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

Parameters **file** (*IO*) – 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: madam.core.Asset, width: int, height: int*) → madam.core.Asset

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: madam.core.Asset, angle: float, expand: bool = False*) → madam.core.Asset

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: madam.core.Asset, from_seconds: float = 0, to_seconds: float = 0*) → madam.core.Asset

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*

5.4 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` (*config: Optional[Mapping[str, Any]] = None*)

Bases: `madam.core.Processor`

Represents a processor that uses Pillow as a backend.

__init__ (*config: Optional[Mapping[str, Any]] = None*) → None

Initializes a new *PillowProcessor*.

Parameters **config** – Mapping with settings.

auto_orient (*asset: madam.core.Asset*) → `madam.core.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: IO*) → bool

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

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

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

Return type bool

convert (*asset: madam.core.Asset, mime_type: Union[madam.mime.MimeType, str], color_space: Optional[str] = None, depth: Optional[int] = None, data_type: Optional[str] = None*) → `madam.core.Asset`

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: madam.core.Asset, x: int, y: int, width: int, height: int*) → `madam.core.Asset`

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: madam.core.Asset, orientation: madam.image.FlipOrientation*) → madam.core.Asset
Creates a new asset whose essence is flipped according to 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: IO*) → madam.core.Asset

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

Parameters **file** (*IO*) – 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: madam.core.Asset, width: int, height: int, mode: madam.image.ResizeMode = <ResizeMode.EXACT: 0>*) → madam.core.Asset

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: madam.core.Asset, angle: float, expand: bool = False*) → madam.core.Asset

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: madam.core.Asset*) → madam.core.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

5.5 madam.mime module

class madam.mime.MimeType(*mediatype*: Union[str, MimeType, None], *subtype*: Optional[str] = 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*: Union[str, MimeType, None], *subtype*: Optional[str] = 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.

5.6 madam.vector module

class madam.vector.SVGMetadataProcessor(*config*: Optional[Mapping[str, Any]] = None)

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__(*config*: Optional[Mapping[str, Any]] = None) → None

Initializes a new `SVGMetadataProcessor`.

Parameters **config** – Mapping with settings.

combine (*file: IO, metadata: Mapping[str, Mapping[KT, VT_co]]*) → IO

Returns a byte stream whose contents represent the specified file where the specified metadata was added.

Parameters

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

Returns file-like object with combined content

Return type IO

formats

The metadata formats which are supported.

Returns supported metadata formats

Return type set[str]

read (*file: IO*) → Mapping[str, Mapping[KT, VT_co]]

Reads the file and returns the metadata.

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

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

Returns Metadata contained in the file

Return type Mapping

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

strip (*file: IO*) → IO

Removes all metadata of the supported type from the specified file.

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

Returns file-like object without metadata

Return type IO

class madam.vector.SVGProcessor (*config: Optional[Mapping[str, Any]] = None*)

Bases: *madam.core.Processor*

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

__init__ (*config: Optional[Mapping[str, Any]] = None*) → None

Initializes a new *SVGProcessor*.

Parameters **config** – Mapping with settings.

can_read (*file: IO*) → bool

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

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

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

Return type bool

read (*file: IO*) → madam.core.Asset

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

Parameters **file** (*IO*) – 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: madam.core.Asset*) → madam.core.Asset

Shrinks the size of an SVG asset.

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

Returns Shrunk vector asset

Return type *Asset*

CHAPTER 6

Index

m

- `madam.core`, [11](#)
- `madam.exif`, [17](#)
- `madam.ffmpeg`, [18](#)
- `madam.image`, [21](#)
- `madam.mime`, [24](#)
- `madam.vector`, [24](#)

Symbols

[__init__\(\)](#) (*madam.core.Asset* method), 11
[__init__\(\)](#) (*madam.core.AssetStorage* method), 11
[__init__\(\)](#) (*madam.core.InMemoryStorage* method), 12
[__init__\(\)](#) (*madam.core.Madam* method), 13
[__init__\(\)](#) (*madam.core.MetadataProcessor* method), 14
[__init__\(\)](#) (*madam.core.OperatorError* method), 15
[__init__\(\)](#) (*madam.core.Pipeline* method), 15
[__init__\(\)](#) (*madam.core.Processor* method), 16
[__init__\(\)](#) (*madam.core.ShelveStorage* method), 16
[__init__\(\)](#) (*madam.core.UnsupportedFormatError* method), 17
[__init__\(\)](#) (*madam.exif.ExifMetadataProcessor* method), 17
[__init__\(\)](#) (*madam.ffmpeg.FFmpegMetadataProcessor* method), 18
[__init__\(\)](#) (*madam.ffmpeg.FFmpegProcessor* method), 19
[__init__\(\)](#) (*madam.image.PillowProcessor* method), 22
[__init__\(\)](#) (*madam.mime.MimeType* method), 24
[__init__\(\)](#) (*madam.vector.SVGMetadataProcessor* method), 24
[__init__\(\)](#) (*madam.vector.SVGProcessor* method), 25

A

[add\(\)](#) (*madam.core.Pipeline* method), 15
[Asset](#) (class in *madam.core*), 11
[AssetStorage](#) (class in *madam.core*), 11
[auto_orient\(\)](#) (*madam.image.PillowProcessor* method), 22

C

[can_read\(\)](#) (*madam.core.Processor* method), 16
[can_read\(\)](#) (*madam.ffmpeg.FFmpegProcessor* method), 19

[can_read\(\)](#) (*madam.image.PillowProcessor* method), 22
[can_read\(\)](#) (*madam.vector.SVGProcessor* method), 25
[clear\(\)](#) (*madam.core.AssetStorage* method), 12
[clear\(\)](#) (*madam.core.InMemoryStorage* method), 12
[clear\(\)](#) (*madam.core.ShelveStorage* method), 16
[combine\(\)](#) (*madam.core.MetadataProcessor* method), 14
[combine\(\)](#) (*madam.exif.ExifMetadataProcessor* method), 17
[combine\(\)](#) (*madam.ffmpeg.FFmpegMetadataProcessor* method), 18
[combine\(\)](#) (*madam.vector.SVGMetadataProcessor* method), 24
[convert\(\)](#) (*madam.ffmpeg.FFmpegProcessor* method), 19
[convert\(\)](#) (*madam.image.PillowProcessor* method), 22
[crop\(\)](#) (*madam.ffmpeg.FFmpegProcessor* method), 20
[crop\(\)](#) (*madam.image.PillowProcessor* method), 22

E

[essence](#) (*madam.core.Asset* attribute), 11
[EXACT](#) (*madam.image.ResizeMode* attribute), 24
[ExifMetadataProcessor](#) (class in *madam.exif*), 17
[extract_frame\(\)](#) (*madam.ffmpeg.FFmpegProcessor* method), 20

F

[FFmpegMetadataProcessor](#) (class in *madam.ffmpeg*), 18
[FFmpegProcessor](#) (class in *madam.ffmpeg*), 19
[FILL](#) (*madam.image.ResizeMode* attribute), 24
[filter\(\)](#) (*madam.core.AssetStorage* method), 12
[filter\(\)](#) (*madam.core.InMemoryStorage* method), 12
[filter\(\)](#) (*madam.core.ShelveStorage* method), 16
[filter_by_tags\(\)](#) (*madam.core.AssetStorage* method), 12

`filter_by_tags()` (*madam.core.InMemoryStorage method*), 12
`filter_by_tags()` (*madam.core.ShelveStorage method*), 16
`FIT` (*madam.image.ResizeMode attribute*), 24
`flip()` (*madam.image.PillowProcessor method*), 22
`FlipOrientation` (*class in madam.image*), 21
`formats` (*madam.core.MetadataProcessor attribute*), 15
`formats` (*madam.exif.ExifMetadataProcessor attribute*), 18
`formats` (*madam.ffmpeg.FFmpegMetadataProcessor attribute*), 18
`formats` (*madam.vector.SVGMetadataProcessor attribute*), 25

G

`get()` (*madam.core.AssetStorage method*), 12
`get()` (*madam.core.InMemoryStorage method*), 13
`get()` (*madam.core.ShelveStorage method*), 16
`get_processor()` (*madam.core.Madam method*), 13

H

`HORIZONTAL` (*madam.image.FlipOrientation attribute*), 21

I

`InMemoryStorage` (*class in madam.core*), 12
`items()` (*madam.core.AssetStorage method*), 12
`items()` (*madam.core.InMemoryStorage method*), 13
`items()` (*madam.core.ShelveStorage method*), 16

K

`keys()` (*madam.core.AssetStorage method*), 12
`keys()` (*madam.core.InMemoryStorage method*), 13
`keys()` (*madam.core.ShelveStorage method*), 17

M

`Madam` (*class in madam.core*), 13
`madam.core` (*module*), 11
`madam.exif` (*module*), 17
`madam.ffmpeg` (*module*), 18
`madam.image` (*module*), 21
`madam.mime` (*module*), 24
`madam.vector` (*module*), 24
`MetadataProcessor` (*class in madam.core*), 14
`MimeType` (*class in madam.mime*), 24

O

`operator()` (*in module madam.core*), 17
`OperatorError`, 15

P

`PillowProcessor` (*class in madam.image*), 21

`Pipeline` (*class in madam.core*), 15
`pop()` (*madam.core.AssetStorage method*), 12
`pop()` (*madam.core.InMemoryStorage method*), 13
`pop()` (*madam.core.ShelveStorage method*), 17
`popitem()` (*madam.core.AssetStorage method*), 12
`popitem()` (*madam.core.InMemoryStorage method*), 13
`popitem()` (*madam.core.ShelveStorage method*), 17
`process()` (*madam.core.Pipeline method*), 15
`Processor` (*class in madam.core*), 15

R

`read()` (*madam.core.Madam method*), 13
`read()` (*madam.core.MetadataProcessor method*), 15
`read()` (*madam.core.Processor method*), 16
`read()` (*madam.exif.ExifMetadataProcessor method*), 18
`read()` (*madam.ffmpeg.FFmpegMetadataProcessor method*), 19
`read()` (*madam.ffmpeg.FFmpegProcessor method*), 20
`read()` (*madam.image.PillowProcessor method*), 23
`read()` (*madam.vector.SVGMetadataProcessor method*), 25
`read()` (*madam.vector.SVGProcessor method*), 25
`resize()` (*madam.ffmpeg.FFmpegProcessor method*), 20
`resize()` (*madam.image.PillowProcessor method*), 23
`ResizeMode` (*class in madam.image*), 23
`rotate()` (*madam.ffmpeg.FFmpegProcessor method*), 21
`rotate()` (*madam.image.PillowProcessor method*), 23

S

`setdefault()` (*madam.core.AssetStorage method*), 12
`setdefault()` (*madam.core.InMemoryStorage method*), 13
`setdefault()` (*madam.core.ShelveStorage method*), 17
`ShelveStorage` (*class in madam.core*), 16
`shrink()` (*madam.vector.SVGProcessor method*), 26
`strip()` (*madam.core.MetadataProcessor method*), 15
`strip()` (*madam.exif.ExifMetadataProcessor method*), 18
`strip()` (*madam.ffmpeg.FFmpegMetadataProcessor method*), 19
`strip()` (*madam.vector.SVGMetadataProcessor method*), 25
`SVGMetadataProcessor` (*class in madam.vector*), 24
`SVGProcessor` (*class in madam.vector*), 25

T

`transpose()` (*madam.image.PillowProcessor*

method), [23](#)

`trim()` (*madam.ffmpeg.FFmpegProcessor method*), [21](#)

U

`UnsupportedFormatError`, [17](#)

`update()` (*madam.core.AssetStorage method*), [12](#)

`update()` (*madam.core.InMemoryStorage method*), [13](#)

`update()` (*madam.core.ShelveStorage method*), [17](#)

V

`values()` (*madam.core.AssetStorage method*), [12](#)

`values()` (*madam.core.InMemoryStorage method*), [13](#)

`values()` (*madam.core.ShelveStorage method*), [17](#)

`VERTICAL` (*madam.image.FlipOrientation attribute*), [21](#)

W

`with_traceback()` (*madam.core.OperatorError method*), [15](#)

`with_traceback()` (*madam.core.UnsupportedFormatError method*), [17](#)

`write()` (*madam.core.Madam method*), [14](#)