Frontmatter Documentation

Release 1.0.0

Chris Amico

Aug 06, 2023

Contents

Jekyll-style YAML front matter offers a useful way to add arbitrary, structured metadata to text documents, regardless of type.

This is a package to load and parse files (or text strings) with YAML front matter.

CHAPTER 1

Install

```
pip install python-frontmatter
```

CHAPTER 2

Usage

```
>>> import frontmatter
```

Load a post from a filename:

```
>>> post = frontmatter.load('tests/yaml/hello-world.txt')
```

Or a file (or file-like object):

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     post = frontmatter.load(f)
```

Or load from text:

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     post = frontmatter.loads(f.read())
```

Access content:

```
>>> print(post.content)
Well, hello there, world.

# this works, too
>>> print(post)
Well, hello there, world.
```

Use metadata (metadata gets proxied as post keys):

```
>>> print(post['title'])
Hello, world!
```

Metadata is a dictionary, with some handy proxies:

```
>>> sorted(post.keys())
['layout', 'title']

>>> from pprint import pprint
>>> post['excerpt'] = 'tl;dr'
>>> pprint(post.metadata)
{'excerpt': 'tl;dr', 'layout': 'post', 'title': 'Hello, world!'}
```

If you don't need the whole post object, use *frontmatter.parse* to return metadata and content separately:

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     metadata, content = frontmatter.parse(f.read())
>>> print(metadata['title'])
Hello, world!
```

Write back to plain text, too:

```
>>> print(frontmatter.dumps(post))
---
excerpt: tl;dr
layout: post
title: Hello, world!
---
Well, hello there, world.
```

Or write to a file (or file-like object):

```
>>> from io import BytesIO
>>> f = BytesIO()
>>> frontmatter.dump(post, f)
>>> print(f.getvalue())
---
excerpt: tl;dr
layout: post
title: Hello, world!
---
Well, hello there, world.
```

2.1 Customizing input and output

By default, `frontmatter` reads and writes YAML metadata. But maybe you don't like YAML. Maybe enjoy writing metadata in JSON, or TOML, or some other exotic markup not yet invented. For this, there are handlers.

This module includes handlers for YAML, JSON and TOML, as well as a *`BaseHandler`* that outlines the basic API and can be subclassed to deal with new formats.

Note: The TOML handler is only available if the toml library is installed.

2.1.1 Handlers

Handlers do most of the underlying work parsing and exporting front matter. When you call *`frontmatter.loads`*, frontmatter first needs to figure out the best handler for the format you're using (YAML, JSON, TOML, etc), then call methods to read or write metadata.

A handler needs to do four things:

- detect whether it can parse the given piece of text
- split front matter from content, returning both as a two-tuple
- parse front matter into a Python dictionary
- export a dictionary back into text

An example:

Calling *frontmatter.load* (or *loads*) with the `handler` argument tells frontmatter which handler to use. The handler instance gets saved as an attribute on the returned post object. By default, calling *frontmatter.dumps* on the post will use the attached handler.

```
>>> import frontmatter
>>> from frontmatter.default_handlers import YAMLHandler, TOMLHandler
>>> post = frontmatter.load('tests/toml/hello-toml.md', handler=TOMLHandler())
>>> post.handler
<frontmatter.default_handlers.TOMLHandler object at 0x...>

>>> print(frontmatter.dumps(post))
+++
test = "tester"
something = "else"
author = "bob"
+++

Title
=====

title2
------

Hello.

Just need three dashes
---

And this shouldn't break.
```

Passing a new handler to *frontmatter.dumps* (or *dump*) changes the export format:

```
>>> print(frontmatter.dumps(post, handler=YAMLHandler()))
---
author: bob
something: else
test: tester
---

Title
=====

title2
------

Hello.

Just need three dashes
---
```

(continues on next page)

(continued from previous page)

```
And this shouldn't break.
```

Changing the attached `handler` on a post has the same effect. Setting `handler` to `None` will default the post back to *YAMLHandler*. These three variations will produce the same export:

```
# set YAML format when dumping, but the old handler attached
>>> t1 = frontmatter.dumps(post, handler=YAMLHandler())
>>> post.handler = YAMLHandler() # set a new handler, changing all future exports
>>> t2 = frontmatter.dumps(post)
>>> post.handler = None # remove handler, defaulting back to YAML
>>> t3 = frontmatter.dumps(post)
>>> t1 == t2 == t3
True
```

All handlers use the interface defined on `BaseHandler`. Each handler needs to know how to:

- split metadata and content, based on a boundary pattern (`handler.split`)
- parse plain text metadata into a Python dictionary (`handler.load`)
- export a dictionary back into plain text (`handler.export`)
- format exported metadata and content into a single string (`handler.format`)

class frontmatter.default_handlers.**BaseHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)

BaseHandler lays out all the steps to detecting, splitting, parsing and exporting front matter metadata.

All default handlers are subclassed from BaseHandler.

detect(*text*)

Decide whether this handler can parse the given `text`, and return True or False.

Note that this is *not* called when passing a handler instance to *frontmatter.load* or *loads*.

export(*metadata*, ***kwargs*)

Turn metadata back into text

format(*post*, ***kwargs*)

Turn a post into a string, used in `frontmatter.dumps`

load(*fm*)

Parse frontmatter and return a dict

split(*text*)

Split text into frontmatter and content

class frontmatter.default_handlers.**YAMLHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)

Load and export YAML metadata. By default, this handler uses YAML's "safe" mode, though it's possible to override that.

class frontmatter.default_handlers.**JSONHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)

Load and export JSON metadata.

Note that changing `START_DELIMITER` or `END_DELIMITER` may break JSON parsing.

class frontmatter.default_handlers.**TOMLHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)

Load and export TOML metadata.

By default, split based on +++.

2.2 API

2.2.1 Reading

frontmatter.**parse**(*text*, *encoding='utf-8'*, *handler=None*, ***defaults*)

Parse text with frontmatter, return metadata and content. Pass in optional metadata defaults as keyword args.

If frontmatter is not found, returns an empty metadata dictionary (or defaults) and original text content.

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     metadata, content = frontmatter.parse(f.read())
>>> print(metadata['title'])
Hello, world!
```

frontmatter.**check**(*fd*, *encoding='utf-8'*)

Check if a file-like object or filename has a frontmatter, return True if exists, False otherwise.

If it contains a frontmatter but it is empty, return True as well.

```
>>> frontmatter.check('tests/yaml/hello-world.txt')
True
```

frontmatter.**checks**(*text*, *encoding='utf-8'*)

Check if a text (binary or unicode) has a frontmatter, return True if exists, False otherwise.

If it contains a frontmatter but it is empty, return True as well.

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     frontmatter.checks(f.read())
True
```

frontmatter.**load**(*fd*, *encoding='utf-8'*, *handler=None*, ***defaults*)

Load and parse a file-like object or filename, return a *post*.

```
>>> post = frontmatter.load('tests/yaml/hello-world.txt')
>>> with open('tests/yaml/hello-world.txt') as f:
...     post = frontmatter.load(f)
```

frontmatter.**loads**(*text*, *encoding='utf-8'*, *handler=None*, ***defaults*)

Parse text (binary or unicode) and return a *post*.

```
>>> with open('tests/yaml/hello-world.txt') as f:
...     post = frontmatter.loads(f.read())
```

2.2.2 Writing

frontmatter.**dump**(*post*, *fd*, *encoding='utf-8'*, *handler=None*, ***kwargs*)

Serialize *post* to a string and write to a file-like object. Text will be encoded on the way out (utf-8 by default).

```
>>> from io import BytesIO
>>> post = frontmatter.load('tests/yaml/hello-world.txt')
>>> f = BytesIO()
>>> frontmatter.dump(post, f)
>>> print(f.getvalue().decode('utf-8'))
---
layout: post
title: Hello, world!
---

Well, hello there, world.
```

```
from io import BytesIO
post = frontmatter.load('tests/yaml/hello-world.txt')
f = BytesIO()
frontmatter.dump(post, f)
print(f.getvalue().decode('utf-8'))
```

```
---
layout: post
title: Hello, world!
---
<BLANKLINE>
Well, hello there, world.
```

frontmatter.**dumps**(*post*, *handler=None*, ***kwargs*)

Serialize a *post* to a string and return text. This always returns unicode text, which can then be encoded.

Passing `handler` will change how metadata is turned into text. A handler passed as an argument will override `post.handler`, with *YAMLHandler* used as a default.

```
>>> post = frontmatter.load('tests/yaml/hello-world.txt')
>>> print(frontmatter.dumps(post))
---
layout: post
title: Hello, world!
---

Well, hello there, world.
```

```
post = frontmatter.load('tests/yaml/hello-world.txt')
print(frontmatter.dumps(post))
```

```
---
layout: post
title: Hello, world!
---

Well, hello there, world.
```

2.2.3 Post objects

class frontmatter.**Post**(*content*, *handler=None*, ***metadata*)

A post contains content and metadata from Front Matter. This is what gets returned by *load* and *loads*. Passing this to *dump* or *dumps* will turn it back into text.

For convenience, metadata values are available as proxied item lookups.

__delitem__(*name*)
Delete a metadata key

__getitem__(*name*)
Get metadata key

__setitem__(*name*, *value*)
Set a metadata key

get(*key*, *default=None*)
Get a key, fallback to default

keys()
Return metadata keys

to_dict()
Post as a dict, for serializing

values()
Return metadata values

2.2.4 Handlers

class frontmatter.default_handlers.**BaseHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)
BaseHandler lays out all the steps to detecting, splitting, parsing and exporting front matter metadata.

All default handlers are subclassed from BaseHandler.

detect(*text*)
Decide whether this handler can parse the given `text`, and return True or False.

Note that this is *not* called when passing a handler instance to *frontmatter.load* or *loads*.

export(*metadata*, ***kwargs*)
Turn metadata back into text

format(*post*, ***kwargs*)
Turn a post into a string, used in `frontmatter.dumps`

load(*fm*)
Parse frontmatter and return a dict

split(*text*)
Split text into frontmatter and content

class frontmatter.default_handlers.**YAMLHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)
Load and export YAML metadata. By default, this handler uses YAML's "safe" mode, though it's possible to override that.

class frontmatter.default_handlers.**JSONHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)
Load and export JSON metadata.

Note that changing `START_DELIMITER` or `END_DELIMITER` may break JSON parsing.

class frontmatter.default_handlers.**TOMLHandler**(*fm_boundary=None*, *start_delimiter=None*, *end_delimiter=None*)

Load and export TOML metadata.

By default, split based on +++.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index