Python-Markups

Release 3.1.3

unknown

Jan 14, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION TO PYTHON-MARKUPS

Python-Markups is a module that provides unified interface for using various markup languages, such as Markdown, reStructuredText, and Textile. It is also possible for clients to create and register their own markup languages.

The output language Python-Markups works with is HTML. Stylesheets and JavaScript sections are supported.

The abstract interface that any markup implements is *AbstractMarkup*.

CHAPTER

TWO

CONTENTS

2.1 API overview

For the basic usage of Python-Markups, one should import some markup class from `markups`, create an instance of that class, and use the *`convert()`* method:

```
>>> import markups
>>> markup = markups.ReStructuredTextMarkup()
>>> markup.convert('*reStructuredText* test').get_document_body()
'<div class="document">\n<p><em>reStructuredText</em> test</p>\n</div>\n'
```

For advanced usage (like dynamically choosing the markup class), one may use one of the functions documented below.

2.1.1 Getting lists of available markups

`markups.`**`get_all_markups`**`()` → List[Type[*markups.abstract.AbstractMarkup*]]

Returns list of all markups (both standard and custom ones)

`markups.`**`get_available_markups`**`()` → List[Type[*markups.abstract.AbstractMarkup*]]

Returns list of all available markups (markups whose *`available()`* method returns True)

2.1.2 Getting a specific markup

`markups.`**`get_markup_for_file_name`**`(`*filename: str*, *return_class: bool = False*`)`

Parameters

- **`filename`** – name of the file
- **`return_class`** – if true, this function will return a class rather than an instance

Returns a markup with *`file_extensions`* attribute containing extension of *filename*, if found, otherwise `None`

```
>>> import markups
>>> markup = markups.get_markup_for_file_name('foo.mkd')
>>> markup.convert('**Test**').get_document_body()
```

(continues on next page)

(continued from previous page)

```
'<p><strong>Test</strong></p>\n'
>>> markups.get_markup_for_file_name('bar.rst', return_class=True)
<class 'markups.restructuredtext.ReStructuredTextMarkup'>
```

markups.**find_markup_class_by_name**(*name: str*) → Optional[Type[*markups.abstract.AbstractMarkup*]]

Returns a markup with *name* attribute matching *name*, if found, otherwise None

```
>>> import markups
>>> markups.find_markup_class_by_name('textile')
<class 'markups.textile.TextileMarkup'>
```

2.1.3 Configuration directory

Some markups can provide configuration files that the user may use to change the behavior.

These files are stored in a single configuration directory.

If XDG_CONFIG_HOME is defined, then the configuration directory is it. Otherwise, it is .config subdirectory in the user's home directory.

2.2 Markup interface

The main class for interacting with markups is *AbstractMarkup*.

However, you shouldn't create direct instances of that class. Instead, use one of the *standard markup classes*.

class markups.abstract.**AbstractMarkup**(*filename: Optional[str] = None*)

Abstract class for markup languages.

Parameters **filename** – optional name of the file

attributes: Dict[int, Any]

various attributes, like links to website and syntax documentation

static available() → bool

Returns

whether the markup is ready for use

(for example, whether the required third-party modules are importable)

convert(*text: str*) → *markups.abstract.ConvertedMarkup*

Returns a ConvertedMarkup instance (or a subclass thereof) containing the markup converted to HTML

default_extension: str

the default file extension

file_extensions: Tuple[str, ...]

indicates which file extensions are associated with the markup

name: str
name of the markup visible to user

When *AbstractMarkup*'s *convert()* method is called it will return an instance of *ConvertedMarkup* or a subclass thereof that provides access to the conversion results.

class markups.abstract.**ConvertedMarkup**(*body: str*, *title: str = ''*, *stylesheet: str = ''*, *javascript: str = ''*)
This class encapsulates the title, body, stylesheet and javascript of a converted document.

Instances of this class are created by *AbstractMarkup.convert()* method, usually it should not be instantiated directly.

get_document_body() → str

Returns the contents of the <body> HTML tag

get_document_title() → str

Returns the document title

get_javascript(*webenv: bool = False*) → str

Returns one or more HTML tags to be inserted into the document <head>.

Parameters **webenv** – if true, the specific markups may optimize the document for being used in the World Wide Web (for example, a remote version of MathJax script can be inserted instead of the local one).

get_stylesheet() → str

Returns the contents of <style type="text/css"> HTML tag

get_whole_html(*custom_headers: str = ''*, *include_stylesheet: bool = True*, *fallback_title: str = ''*, *webenv: bool = False*) → str

Returns the full contents of the HTML document (unless overridden this is a combination of the previous methods)

Parameters

- **custom_headers** – custom HTML to be inserted into the document <head>
- **include_stylesheet** – if false, the stylesheet will not be included in the document <head>
- **fallback_title** – when impossible to get the <title> from the document, this string can be used as a fallback
- **webenv** – like in *get_javascript()* above

2.3 Built-in markups

These markups are available by default:

2.3.1 Markdown markup

Markdown markup uses Python-Markdown as a backend (version 2.6 or later is required).

There are several ways to enable Python-Markdown extensions.

- List extensions in a file named `markdown-extensions.yaml` or `markdown-extensions.txt` in the *configuration directory*. The extensions will be automatically applied to all documents.
- If `markdown-extensions.yaml` or `markdown-extensions.txt` is placed into working directory, all documents in that directory will get extensions that are listed in that file.
- If first line of a document contains "`Required extensions: ext1 ext2 ...`", that list will be applied to a document.
- Finally, one can programmatically pass list of extension names to *markups.MarkdownMarkup* constructor.

The YAML file should be a list of extensions, possibly with configuration options, for example:

```
- smarty:
    substitutions:
      left-single-quote: "&sbquo;"
      right-single-quote: "&lsquo;"
    smart_dashes: False
- toc:
    permalink: True
    separator: "_"
    toc_depth: 3
- sane_lists
```

Or using a JSON-like syntax:

```
["smarty", "sane_lists"]
```

YAML support works only when the PyYAML module is installed.

The txt file is a simple list of extensions, separated by newlines. Lines starting with # are treated as comments and ignored. It is possible to specify string options in brackets, for example:

```
toc(title=Contents)
sane_lists
```

The same syntax to specify options works in the `Required extensions` line. You can put it into a comment to make it invisible in the output:

```
<!-- Required extensions: toc(title=Contents) sane_lists -->
```

The Math Markdown extension is enabled by default. This extension supports a syntax for LaTeX-style math formulas (powered by MathJax). The delimiters are:

Inline math	Standalone math
`$...$`[1]	`$$...$$`
`\(...\)`	`\[...\]`

The Python-Markdown Extra set of extensions is enabled by default. To disable it, one can enable virtual `remove_extra` extension (which also completely disables LaTeX formulas support).

The default file extension associated with Markdown markup is `.mkd`, though many other extensions (including `.md` and `.markdown`) are supported as well.

class markups.**MarkdownMarkup**(*filename=None*, *extensions=None*)
Markup class for Markdown language. Inherits *AbstractMarkup*.

Parameters **extensions** (*list*) – list of extension names

2.3.2 reStructuredText markup

This markup provides support for reStructuredText language (the language this documentation is written in). It uses Docutils Python module.

The file extension associated with reStructuredText markup is `.rst`.

class markups.**ReStructuredTextMarkup**(*filename=None*, *settings_overrides=None*)
Markup class for reStructuredText language. Inherits *AbstractMarkup*.

Parameters **settings_overrides** (*dict*) – optional dictionary of overrides for the Docutils settings

2.3.3 Textile markup

This markup provides support for Textile language. It uses python-textile module.

The file extension associated with Textile markup is `.textile`.

class markups.**TextileMarkup**(*filename=None*)
Markup class for Textile language. Inherits *AbstractMarkup*.

2.4 Custom Markups

2.4.1 Registering the markup module

A third-party markup is a Python module that can be installed the usual way.

To register your markup class with PyMarkups, make it inherit from *AbstractMarkup*, and add that class to your module's `entry_points`, in the "pymarkups" entry point group.

For example:

```
setup(
    ...
    entry_points={
        'pymarkups': [
            'mymarkup = mymodule:MyMarkupClass',
        ],
    },
    ...
)
```

[1] To enable single-dollar-sign delimiter, one should add `mdx_math(enable_dollar_delimiter=1)` to the extensions list.

Or using the declarative syntax in `setup.cfg`:

```
[options.entry_points]
pymarkups =
    mymarkup = mymodule:MyMarkupClass
```

See the setuptools documentation on entry points for details.

To check if the module was found by Python-Markups, one can check if the module is present in return value of *get_all_markups()* function.

Changed in version 3.0: The custom markups should be registered using the entry points mechanism, the `pymarkups.txt` file is no longer supported.

2.4.2 Importing third-party modules

A markup must not directly import any third party Python module it uses at file level. Instead, it should check the module availability in *available()* static method.

That method can try to import the needed modules, and return `True` in case of success, and `False` in case of failure.

2.4.3 Implementing methods

Any markup must inherit from *AbstractMarkup*.

Third-party markups must implement *AbstractMarkup*'s *convert()* method, which must perform the time-consuming part of markup conversion and return a newly constructed instance of (a subclass of) *ConvertedMarkup*.

ConvertedMarkup encapsulates the title, body, stylesheet and javascript of a converted document. Of these only the body is required during construction, the others default to an empty string. If additional markup-specific state is required to implement *ConvertedMarkup*, a subclass can be defined and an instance of it returned from *convert()* instead.

2.5 Python-Markups changelog

This changelog only lists the most important changes that happened in Python-Markups. Please see the Git log for the full list of changes.

2.5.1 Version 3.1.3, 2021-11-21

- Fixed logic to load extensions file when PyYAML module is not available (issue #16, thanks foxB612 for the bug report).
- Made the tests pass with docutils 0.18.

2.5.2 Version 3.1.2, 2021-09-06

- Incompatible change: Python 3.6 is no longer supported.
- Fixed replacing Markdown extensions in document.
- Fixed crash when using TOC backrefs in reStructuredText (issue #14, thanks Hrissimir for the patch).

2.5.3 Version 3.1.1, 2021-03-05

- The reStructuredText markup now includes line numbers information in `data-posmap` attributes.
- The reStructuredText markup now uses only `minimal.css` stylesheet (not `plain.css` anymore).
- Added support for the upcoming docutils 0.17 release to the tests.

2.5.4 Version 3.1.0, 2021-01-31

Incompatible changes:

- Python versions older than 3.6 are no longer supported.

Other changes:

- Instead of `pkg_resources`, `importlib.metadata` is now used.
- For Markdown markup, `markdown-extensions.yaml` files are now supported in addition to `markdown-extensions.txt` files.
- Type annotations were added for public API.
- The reStructuredText markup no longer raises exceptions for invalid markup.
- MathJax v3 is now supported in addition to v2. Also, the Arch Linux mathjax packages are now supported (issue #4).
- Added Pygments CSS support for the `pymdownx.highlight` Markdown extension.

2.5.5 Version 3.0.0, 2018-05-03

Incompatible changes:

- The deprecated AbstractMarkup API has been removed.
- Python 3.2 is no longer supported.
- The output now uses HTML5 instead of HTML4.
- The custom markups are now registered with entry points.
- The `get_custom_markups()` method has been removed.
- New required dependency: python-markdown-math.

Other changes:

- The upcoming Python-Markdown 3.x release is now supported.

2.5.6 Version 2.0.1, 2017-06-24

- The new MathJax CDN is used, the old one will be shut down soon.
- When using MathJax with Markdown, the AMSmath and AMSsymbols extensions are now enabled.

2.5.7 Version 2.0.0, 2016-05-09

Incompatible changes:

- Changed the API of pymarkups to clearly separate the conversion step from access to the various elements of the result. The old API is deprecated and will be removed in a future release. Please see the documentation for details on using the new API.
- The reStructuredText markup now includes document title and subtitle in the HTML body.

Other changes:

- Added a `markup2html.py` reference script to show API usage.
- Improved support for specifying Markdown extensions in the document.

2.5.8 Version 1.0.1, 2015-12-22

- The Textile markup now uses the recommended python-textile API.
- Fixed warnings during installation.
- Python-Markdown Math extension updated to the latest version.

2.5.9 Version 1.0, 2015-12-13

- Web module removed, as ReText no longer needs it.
- Textile markup updated to work with the latest version of Python-Textile module.
- The setup script now uses setuptools when it is available.
- Testsuite and documentation improvements.

2.5.10 Version 0.6.3, 2015-06-16

- No-change re-upload with fixed tarball and changelog.

2.5.11 Version 0.6.2, 2015-06-09

- Markdown markup: fixed detection of codehilite extension with options.
- Added a warning about deprecation of the markups.web module.

2.5.12 Version 0.6.1, 2015-04-19

- PyMarkups now uses warnings system instead of printing messages to stderr.
- Improvements to Markdown markup:
  - Fixed parsing math that contains nested environments (thanks to Gautam Iyer for the patch).
  - Fixed crash on extensions names starting with dot.
- Miscellaneous fixes.

2.5.13 Version 0.6, 2015-01-25

Incompatible changes:

- Custom markups are now normal Python modules.
- Web module no longer supports Python 2.x.

Other changes:

- Refactor the code related to Markdown extensions to make it work with upcoming Python-Markdown releases.
- MathJax extension is now in a separate module.

2.5.14 Version 0.5.2, 2014-11-05

- Fixed loading of Markdown extensions with options.

2.5.15 Version 0.5.1, 2014-09-16

- Fixed Markdown markup crash on empty files.
- Include documentation in the tarballs.
- Testsuite improvements.

2.5.16 Version 0.5, 2014-07-25

- Improvements to Markdown markup:
  - All math delimeters except `$...$` are now enabled by default.
  - `remove_extra` extension now disables formulas support.
  - It is now possible to specify required extensions in the first line of the file.
- Add Sphinx documentation.

2.5.17 Version 0.4, 2013-11-30

- Add Textile markup.
- reStructuredText markup now supports file names and settings overrides.
- Web module now raises WebUpdateError when updating fails.

2.5.18 Version 0.3, 2013-07-25

- MathJax support in Markdown has been improved and no longer relies on tex2jax extension.
- It is now possible to pass extensions list to MarkdownMarkup constructor.
- Pygments style is now configurable.
- Testsuite improvements.

2.5.19 Version 0.2.3, 2012-11-02

- Fix support for custom working directory in web module.
- Bug fixes in Markdown module and tests.

2.5.20 Version 0.2.2, 2012-10-02

- Re-written math support for Markdown.
- Add tests to the tarball.
- Add example template for web module.
- Bug fixes in Markdown and web modules.

2.5.21 Version 0.2.1, 2012-09-09

- Add caching support, to speed up get_document_body function.
- Add testsuite.
- Fix some bugs in markdown module.

2.5.22 Version 0.2, 2012-09-04

- Initial release.

CHAPTER

THREE

LINKS

- Python-Markups source code is hosted on GitHub.
- You can get the source tarball from PyPI.
- It is also packaged in Debian.

INDEX