py-gfm Documentation

Release 0.1.3

The Dart Team, Alexandre Macabies

August 20, 2016

Contents

This is an implementation of GitHub-Flavored Markdown written as an extension to the Python Markdown library. It aims for maximal compatibility with GitHub's rendering.

CHAPTER 1

Installation

```
pip install py-gfm
```

CHAPTER 2

Quick start

2.1 All-in-one extension

```
import markdown
from mdx_gfm import GithubFlavoredMarkdownExtension

source = """
Hello, *world*! This is a ~~good~~marvelous day!
Here is an auto link: https://example.org/

Le me introduce you to [task lists] (https://github.com/blog/1375-task-lists-in-gfm-issues-pulls-comm

- [ ] eggs
- [x] milk

You can also have fenced code blocks:

```
import this
```
"""

# Direct conversion
html = markdown.markdown(source,
                         extensions=[GithubFlavoredMarkdownExtension()])

# Factory-like
md = markdown.Markdown(extensions=[GithubFlavoredMarkdownExtension()])
html = md.convert(source)
```

2.2 À la carte

```
import markdown
from gfm import AutolinkExtension, TaskListExtension

html = markdown.markdown(source,
                         extensions=[AutolinkExtension(),
                                     TaskListExtension(max_depth=2)])
```

CHAPTER 3

Available extensions

3.1 gfm.autolink – Turn URLs into links

The *gfm.autolink* module provides an extension that turns all raw URLs into marked-up links.

This is based on the web-only URL regex by John Gruber (public domain).

This regex seems to line up pretty closely with GitHub's URL matching. Two cases were identified where they differ. In both cases, the regex were slightly modified to bring it in line with GitHub's parsing:

- GitHub accepts FTP-protocol URLs;
- GitHub only accepts URLs with protocols or `www.`, whereas Gruber's regex accepts things like `foo.com/bar`.

3.1.1 Typical usage

```
import markdown
from gfm import AutolinkExtension

print(markdown.markdown("I love this http://example.org/ check it out",
                        extensions=[AutolinkExtension()]))
```

```
<p>I love this <a href="http://example.org/">http://example.org/</a> check it out</p>
```

class gfm.autolink.**AutolinkExtension**(**args*, ***kwargs*)

Bases: markdown.extensions.Extension

An extension that turns URLs into links.

getConfig(*key*, *default=u''*)

Return a setting for the given key or an empty string.

getConfigInfo()

Return all config descriptions as a list of tuples.

getConfigs()

Return all configs settings as a dict.

setConfig(*key*, *value*)

Set a config setting for *key* with the given *value*.

setConfigs(*items*)

Set multiple config settings given a dict or list of tuples.

3.2 gfm.automail – Turn email addresses into links

The *gfm.automail* module provides an extension that turns all raw email addresses into marked-up links.

3.2.1 Typical usage

```
import markdown
from gfm import AutomailExtension

print(markdown.markdown("You can mail me at foo@example.org for more info",
                        extensions=[AutomailExtension()]))
```

```
<p>You can mail me at <a href="mailto:foo@example.org">foo@example.org</a> for more info</p>
```

class gfm.automail.**AutomailExtension**(**args*, ***kwargs*)
Bases: markdown.extensions.Extension

An extension that turns email addresses into links.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

3.3 gfm.hidden_hilite – Fenced code blocks with no highlighting

The *gfm.hidden_hilite* module provides an extension that allows the use of fenced code blocks without adding syntax highlighting or line numbers.

3.3.1 Typical usage

```
import markdown
from gfm import HiddenHiliteExtension

print(markdown.markdown("```\nimport this\nprint('foo')\n```",
                        extensions=[HiddenHiliteExtension()]))
```

```
<p><code>import this
print('foo')</code></p>
```

class gfm.hidden_hilite.**HiddenHiliteExtension**(**args*, ***kwargs*)
Bases: markdown.extensions.codehilite.CodeHiliteExtension

A subclass of CodeHiliteExtension that doesn't highlight on its own.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

3.4 gfm.semi_sane_lists – GitHub-like list parsing

The *gfm.semi_sane_lists* module provides an extension that causes lists to be treated the same way GitHub does.

Like the `sane_lists` extension, GitHub considers a list to end if it's separated by multiple newlines from another list of a different type. Unlike the `sane_lists` extension, GitHub will mix list types if they're not separated by multiple newlines.

GitHub also recognizes lists that start in the middle of a paragraph. This is currently not supported by this extension, since the Python parser has a deeply-ingrained belief that blocks are always separated by multiple newlines.

3.4.1 Typical usage

```
import markdown
from gfm import SemiSaneListExtension

print(markdown.markdown("""
- eggs
- milk

1. mix
2. stew
""", extensions=[SemiSaneListExtension()]))
```

```
<ul>
<li>eggs</li>
<li>milk</li>
</ul>
<ol>
<li>mix</li>
<li>stew</li>
</ol>
```

class gfm.semi_sane_lists.**SemiSaneListExtension**(**args*, ***kwargs*)
Bases: markdown.extensions.Extension

An extension that causes lists to be treated the same way GitHub does.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

3.5 gfm.spaced_link – Links with optional whitespace

The *gfm.spaced_link* module provides an extension that supports links and images with additional whitespace.

GitHub's Markdown engine allows links and images to have whitespace – including a single newline – between the first set of brackets and the second (e.g. `[text] (href)`). This extension adds such support.

3.5.1 Typical usage

```
import markdown
from gfm import SpacedLinkExtension

print(markdown.markdown("Check out [this link] (http://example.org/)!",
                        extensions=[SpacedLinkExtension()]))
```

```
<p>Check out <a href="http://example.org/">this link</a>!</p>
```

class gfm.spaced_link.**SpacedLinkExtension**(**args*, ***kwargs*)
Bases: markdown.extensions.Extension

An extension that supports links and images with additional whitespace.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

3.6 gfm.strikethrough – Strike-through support

The *gfm.strikethrough* module provides GitHub-like syntax for strike-through text, that is text between double tildes: `some ~~strike-through'ed~~ text`

3.6.1 Typical usage

```
import markdown
from gfm import StrikethroughExtension

print(markdown.markdown("I ~~like~~ love you!",
                        extensions=[StrikethroughExtension()]))
```

```
<p>I <del>like</del> love you!</p>
```

class gfm.strikethrough.**StrikethroughExtension**(**args*, ***kwargs*)

Bases: markdown.extensions.Extension

An extension that adds support for strike-through text between two ~~.

getConfig(*key*, *default=u''*)

Return a setting for the given key or an empty string.

getConfigInfo()

Return all config descriptions as a list of tuples.

getConfigs()

Return all configs settings as a dict.

setConfig(*key*, *value*)

Set a config setting for *key* with the given *value*.

setConfigs(*items*)

Set multiple config settings given a dict or list of tuples.

3.7 gfm.tasklist – Task list support

The *gfm.tasklist* module provides GitHub-like support for task lists. Those are normal lists with a checkbox-like syntax at the beginning of items that will be converted to actual checkbox inputs. Nested lists are supported.

Example syntax:

```
- [x] milk
- [ ] eggs
- [x] chocolate
- [ ] if possible:
    1. [ ] solve world peace
    2. [ ] solve world hunger
```

Note: GitHub has support for updating the Markdown source text by toggling the checkbox (by clicking on it). This is not supported by this extension, as it requires server-side processing that is out of scope of a Markdown parser.

3.7.1 Available configuration options

Name	Type	Default	Description
`unordered`	bool	`True`	Set to `False` to disable parsing of unordered lists.
`ordered`	bool	`True`	Set to `False` to disable parsing of ordered lists.
`max_depth`	integer	∞	Set to a positive integer to stop parsing nested task lists that are deeper than this limit.
`list_attrs`	dict, callable	`{}`	Attributes to be added to the `<ul>` or `<ol>` element containing the items.
`item_attrs`	dict, callable	`{}`	Attributes to be added to the `<li>` element containing the checkbox. See *Item attributes*.
`checkbox_attrs`	dict, callable	`{}`	Attributes to be added to the checkbox element. See *Checkbox attributes*.

List attributes

If option `list_attrs` is a *dict*, the key-value pairs will be applied to the `<ul>` (resp. `<ol>`) unordered (resp. ordered) list element, that is the parent element of the `<li>` elements.

> **Warning:** These attributes are applied to all nesting levels of lists, that is, to both the root lists and their potential sub-lists, recursively.
> You can control this behavior by using a *callable* instead (see below).

If option `list_attrs` is a *callable*, it should be a function that respects the following prototype:

```
def function(list, depth: int) -> dict:
```

where:

- `list` is the `<ul>` or `<ol>` element;
- `depth` is the depth of this list relative to its root list (root lists have a depth of 1).

The returned *dict* items will be applied as HTML attributes to the list element.

Note: Thanks to this feature, you could apply attributes to root lists only. Take this code sample:

```
import markdown
from gfm import TaskListExtension

def list_attr_cb(list, depth):
    if depth == 1:
        return {'class': 'tasklist'}
    return {}

tl_ext = TaskListExtension(list_attrs=list_attr_cb)

print(markdown.markdown("""
- [x] some thing
- [ ] some other
    - [ ] sub thing
    - [ ] sub other
""", extensions=[tl_ext]))
```

In this example, only the root list will have the `tasklist` class attribute, not the one containing "sub" items.

Item attributes

If option `item_attrs` is a *dict*, the key-value pairs will be applied to the `<li>` element as its HTML attributes.

Example:

```
item_attrs={'class': 'list-item'}
```

will result in:

```
<li class="list-item">...</li>
```

If option `item_attrs` is a *callable*, it should be a function that respects the following prototype:

```
def function(parent, element, checkbox) -> dict:
```

where:

- `parent` is the `<li>` parent element;
- `element` is the `<li>` element;
- `checkbox` is the generated `<input type="checkbox">` element.

The returned *dict* items will be applied as HTML attributes to the `<li>` element containing the checkbox.

Checkbox attributes

If option `checkbox_attrs` is a *dict*, the key-value pairs will be applied to the `<input type="checkbox">` element as its HTML attributes.

Example:

```
checkbox_attrs={'class': 'list-cb'}
```

will result in:

```
<li><input type="checkbox" class="list-cb"> ...</li>
```

If option `checkbox_attrs` is a *callable*, it should be a function that respects the following prototype:

```
def function(parent, element) -> dict:
```

where:

- `parent` is the `<li>` parent element;
- `element` is the `<li>` element.

The returned *dict* items will be applied as HTML attributes to the checkbox element.

3.7.2 Typical usage

```
import markdown
from gfm import TaskListExtension

print(markdown.markdown("""
- [x] milk
- [ ] eggs
- [x] chocolate
- not a checkbox
""", extensions=[TaskListExtension()]))
```

```
<ul>
<li><input checked="checked" disabled="disabled" type="checkbox" /> milk</li>
<li><input disabled="disabled" type="checkbox" /> eggs</li>
<li><input checked="checked" disabled="disabled" type="checkbox" /> chocolate</li>
<li>not a checkbox</li>
</ul>
```

class gfm.tasklist.**TaskListExtension**(**args*, ***kwargs*)

Bases: markdown.extensions.Extension

An extension that supports GitHub task lists. Both ordered and unordered lists are supported and can be separately enabled. Nested lists are supported.

Example:

```
- [x] milk
- [ ] eggs
- [x] chocolate
- [ ] if possible:
    1. [ ] solve world peace
    2. [ ] solve world hunger
```

getConfig(*key*, *default=u''*)

Return a setting for the given key or an empty string.

getConfigInfo()

Return all config descriptions as a list of tuples.

getConfigs()

Return all configs settings as a dict.

setConfig(*key*, *value*)

Set a config setting for *key* with the given *value*.

setConfigs(*items*)

Set multiple config settings given a dict or list of tuples.

CHAPTER 4

Modules

4.1 gfm – Base module for GitHub-Flavored Markdown

class gfm.**AutolinkExtension**(**args*, ***kwargs*)
An extension that turns URLs into links.

class gfm.**AutomailExtension**(**args*, ***kwargs*)
An extension that turns email addresses into links.

class gfm.**HiddenHiliteExtension**(**args*, ***kwargs*)
A subclass of CodeHiliteExtension that doesn't highlight on its own.

class gfm.**SemiSaneListExtension**(**args*, ***kwargs*)
An extension that causes lists to be treated the same way GitHub does.

class gfm.**SpacedLinkExtension**(**args*, ***kwargs*)
An extension that supports links and images with additional whitespace.

class gfm.**StrikethroughExtension**(**args*, ***kwargs*)
An extension that adds support for strike-through text between two ~~.

class gfm.**TaskListExtension**(**args*, ***kwargs*)
An extension that supports GitHub task lists. Both ordered and unordered lists are supported and can be separately enabled. Nested lists are supported.

Example:

```
- [x] milk
- [ ] eggs
- [x] chocolate
- [ ] if possible:
    1. [ ] solve world peace
    2. [ ] solve world hunger
```

4.2 mdx_gfm – Full extension for GFM (comments, issues)

An extension that is as compatible as possible with GitHub-flavored Markdown (GFM).

This extension aims to be compatible with the standard GFM that GitHub uses for comments and issues. It has all the extensions described in the GFM documentation, except for intra-GitHub links to commits, repositories, and issues.

Note that Markdown-formatted gists and files (including READMEs) on GitHub use a slightly different variant of GFM. For that, use *mdx_partial_gfm.PartialGithubFlavoredMarkdownExtension*.

class mdx_gfm.**GithubFlavoredMarkdownExtension**(**args*, ***kwargs*)
Bases: *mdx_partial_gfm.PartialGithubFlavoredMarkdownExtension*

An extension that is as compatible as possible with GitHub-flavored Markdown (GFM).

This extension aims to be compatible with the standard GFM that GitHub uses for comments and issues. It has all the extensions described in the GFM documentation, except for intra-GitHub links to commits, repositories, and issues.

Note that Markdown-formatted gists and files (including READMEs) on GitHub use a slightly different variant of GFM. For that, use *mdx_partial_gfm.PartialGithubFlavoredMarkdownExtension*.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

4.3 mdx_partial_gfm – Partial extension for GFM (READMEs, wiki)

An extension that is as compatible as possible with GitHub-flavored Markdown (GFM).

This extension aims to be compatible with the variant of GFM that GitHub uses for Markdown-formatted gists and files (including READMEs). This variant seems to have all the extensions described in the GFM documentation, except:

- Newlines in paragraphs are not transformed into `br` tags.
- Intra-GitHub links to commits, repositories, and issues are not supported.

If you need support for features specific to GitHub comments and issues, please use *mdx_gfm.GithubFlavoredMarkdownExtension*.

class mdx_partial_gfm.**PartialGithubFlavoredMarkdownExtension**(**args*, ***kwargs*)
Bases: `markdown.extensions.Extension`

An extension that is as compatible as possible with GitHub-flavored Markdown (GFM).

This extension aims to be compatible with the variant of GFM that GitHub uses for Markdown-formatted gists and files (including READMEs). This variant seems to have all the extensions described in the GFM documentation, except:

•Newlines in paragraphs are not transformed into `br` tags.

•Intra-GitHub links to commits, repositories, and issues are not supported.

If you need support for features specific to GitHub comments and issues, please use *mdx_gfm.GithubFlavoredMarkdownExtension*.

getConfig(*key*, *default=u''*)
Return a setting for the given key or an empty string.

getConfigInfo()
Return all config descriptions as a list of tuples.

getConfigs()
Return all configs settings as a dict.

setConfig(*key*, *value*)
Set a config setting for *key* with the given *value*.

setConfigs(*items*)
Set multiple config settings given a dict or list of tuples.

CHAPTER 5

Supported features

- Fenced code blocks
- Literal line breaks
- Tables
- Hyperlink parsing (`http`, `https`, `ftp`, `email` and `www` subdomains)
- Code highlighting (dummy, no actual syntactic coloration as-is)
- Mixed-style lists with no separation
- Links and images with whitespace
- Strikethrough
- Task lists

CHAPTER 6

Unsupported features

This implementation does not support all of GFM features.

6.1 By design

- Link to commits, issues, pull requests and user profiles: this is application specific. Feel free to subclass the provided classes to implement your own logic.

6.2 Planned

- Horizontal rules
- Emojis

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

g

m

A

G

H

M

P

S

T