py-gfm Documentation

Release 1.0.0

The Dart Team, Alexandre Macabies

Jul 26, 2020

Contents

1 Installation 3

2 Quick start 5

3 Available extensions 7

4 Modules 17

5 Supported features 21

6 Unsupported features 23

7 Indices and tables 25

Python Module Index 27

Index 29

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-comments):

- [ ] eggs
- [x] milk

You can also have fenced code blocks:

```python
import this
```
"""

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

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

# By module name (not recommended if you need custom configs):
html = markdown.markdown(source, extensions=['mdx_gfm'])
```

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**(***kwargs*)

Bases: markdown.extensions.Extension

An extension that turns URLs into links.

extendMarkdown(*md*)

Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

class gfm.autolink.**AutolinkPattern**(*pattern*, *md=None*)
Bases: markdown.inlinepatterns.Pattern

getCompiledRegExp()
Return a compiled regular expression.

handleMatch(*m*)
Return a ElementTree element from the given match.

Subclasses should override this method.

Keyword arguments:

- m: A re match object containing a match of the pattern.

type()
Return class name, to define pattern type

unescape(*text*)
Return unescaped text given text with an inline placeholder.

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**(***kwargs*)
Bases: markdown.extensions.Extension

An extension that turns email addresses into links.

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

class gfm.automail.**AutomailPattern**(*pattern*, *md=None*)
Bases: markdown.inlinepatterns.Pattern

getCompiledRegExp()
Return a compiled regular expression.

handleMatch(*m*)
Return a ElementTree element from the given match.

Subclasses should override this method.

Keyword arguments:

- m: A re match object containing a match of the pattern.

type()
Return class name, to define pattern type

unescape(*text*)
Return unescaped text given text with an inline placeholder.

3.3 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.

3.3.1 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**(***kwargs*)

Bases: markdown.extensions.Extension

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

extendMarkdown(*md*)

Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

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.

class gfm.semi_sane_lists.**SemiSaneOListProcessor**(*parser*)

Bases: markdown.blockprocessors.OListProcessor

detab(*text*)

Remove a tab from the front of each line of the given text.

get_items(*block*)

Break a block into list items.

lastChild(*parent*)
Return the last child of an etree element.

looseDetab(*text*, *level=1*)
Remove a tab from front of lines but allowing dedented lines.

run(*parent*, *blocks*)
Run processor. Must be overridden by subclasses.

When the parser determines the appropriate type of a block, the parser will call the corresponding processor's `run` method. This method should parse the individual lines of the block and append them to the etree.

Note that both the `parent` and `etree` keywords are pointers to instances of the objects which should be edited in place. Each processor must make changes to the existing objects as there is no mechanism to return new/different objects to replace them.

This means that this method should be adding SubElements or adding text to the parent, and should remove (`pop`) or add (`insert`) items to the list of blocks.

Keywords:

- `parent`: A etree element which is the parent of the current block.
- `blocks`: A list of all remaining blocks of the document.

test(*parent*, *block*)
Test for block type. Must be overridden by subclasses.

As the parser loops through processors, it will call the `test` method on each to determine if the given block of text is of that type. This method must return a boolean `True` or `False`. The actual method of testing is left to the needs of that particular block type. It could be as simple as `block.startswith(some_string)` or a complex regular expression. As the block type may be different depending on the parent of the block (i.e. inside a list), the parent etree element is also provided and may be used as part of the test.

Keywords:

- `parent`: A etree element which will be the parent of the block.
- **`block`: A block of text from the source which has been split at** blank lines.

class gfm.semi_sane_lists.**SemiSaneUListProcessor**(*parser*)
Bases: `markdown.blockprocessors.UListProcessor`

detab(*text*)
Remove a tab from the front of each line of the given text.

get_items(*block*)
Break a block into list items.

lastChild(*parent*)
Return the last child of an etree element.

looseDetab(*text*, *level=1*)
Remove a tab from front of lines but allowing dedented lines.

run(*parent*, *blocks*)
Run processor. Must be overridden by subclasses.

When the parser determines the appropriate type of a block, the parser will call the corresponding processor's `run` method. This method should parse the individual lines of the block and append them to the etree.

Note that both the `parent` and `etree` keywords are pointers to instances of the objects which should be edited in place. Each processor must make changes to the existing objects as there is no mechanism to return new/different objects to replace them.

This means that this method should be adding SubElements or adding text to the parent, and should remove (`pop`) or add (`insert`) items to the list of blocks.

Keywords:

- `parent`: A etree element which is the parent of the current block.
- `blocks`: A list of all remaining blocks of the document.

`test` (*parent*, *block*)
Test for block type. Must be overridden by subclasses.

As the parser loops through processors, it will call the `test` method on each to determine if the given block of text is of that type. This method must return a boolean `True` or `False`. The actual method of testing is left to the needs of that particular block type. It could be as simple as `block.startswith(some_string)` or a complex regular expression. As the block type may be different depending on the parent of the block (i.e. inside a list), the parent etree element is also provided and may be used as part of the test.

Keywords:

- `parent`: A etree element which will be the parent of the block.
- **`block`: A block of text from the source which has been split at** blank lines.

3.4 `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`

3.4.1 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`**(***kwargs*)
Bases: `markdown.extensions.Extension`

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

`extendMarkdown`(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

3.5.1 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.

3.5.2 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**(***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
```

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

class gfm.tasklist.**TaskListProcessor**(*ext*)
Bases: markdown.treeprocessors.Treeprocessor

run(*root*)
Subclasses of Treeprocessor should implement a *run* method, which takes a root ElementTree. This method can return another ElementTree object, and the existing root ElementTree will be replaced, or it can modify the current tree and return None.

CHAPTER 4

Modules

4.1 gfm – Base module for GitHub-Flavored Markdown

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

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

class gfm.**StandaloneFencedCodeExtension**(***kwargs*)

extendMarkdown(*md*)
Add FencedBlockPreprocessor to the Markdown instance.

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

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

class gfm.**TaskListExtension**(***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
```

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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**(***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*.

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

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

extendMarkdown(*md*)
Add the various proccesors and patterns to the Markdown Instance.

This method must be overriden by every extension.

Keyword arguments:

- md: The Markdown instance.
- md_globals: Global variables in the markdown module namespace.

getConfig(*key*, *default=''*)
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 for code blocks if Pygments is available
- Mixed-style lists with no separation
- Strikethrough
- Task lists

CHAPTER 6

Unsupported features

This implementation does not support all of GFM features and has known differences in how rendering is done.

- By design, link to commits, issues, pull requests and user profiles are not supported since this is application specific. Feel free to subclass the provided classes to implement your own logic.
- There is no emoji support.
- There is no horizontal rule (`---` ie. `<hr>`) support.
- Nested lists are not behaving exactly like GitHub's: issue #10.
- Contrary to GitHub, only double-tilde'd text renders strikethrough, not single-tile'd: issue #14.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

gfm, 17
gfm.autolink, 7
gfm.automail, 8
gfm.semi_sane_lists, 9
gfm.strikethrough, 12
gfm.tasklist, 13

m

mdx_gfm, 18
mdx_partial_gfm, 19

Index

A

AutolinkExtension (*class in gfm*), 17
AutolinkExtension (*class in gfm.autolink*), 7
AutolinkPattern (*class in gfm.autolink*), 8
AutomailExtension (*class in gfm*), 17
AutomailExtension (*class in gfm.automail*), 8
AutomailPattern (*class in gfm.automail*), 9

D

detab() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 10
detab() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 11

E

extendMarkdown() (*gfm.autolink.AutolinkExtension method*), 7
extendMarkdown() (*gfm.AutolinkExtension method*), 17
extendMarkdown() (*gfm.automail.AutomailExtension method*), 9
extendMarkdown() (*gfm.AutomailExtension method*), 17
extendMarkdown() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
extendMarkdown() (*gfm.SemiSaneListExtension method*), 17
extendMarkdown() (*gfm.StandaloneFencedCodeExtension method*), 18
extendMarkdown() (*gfm.strikethrough.StrikethroughExtension method*), 12
extendMarkdown() (*gfm.StrikethroughExtension method*), 18
extendMarkdown() (*gfm.tasklist.TaskListExtension method*), 16
extendMarkdown() (*gfm.TaskListExtension method*), 18
extendMarkdown() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19
extendMarkdown() (*mdx_partial_gfm.PartialGithubFlavoredMarkdow method*), 19

G

get_items() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 10
get_items() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 11
getCompiledRegExp() (*gfm.autolink.AutolinkPattern method*), 8
getCompiledRegExp() (*gfm.automail.AutomailPattern method*), 9
getConfig() (*gfm.autolink.AutolinkExtension method*), 8
getConfig() (*gfm.automail.AutomailExtension method*), 9
getConfig() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
getConfig() (*gfm.strikethrough.StrikethroughExtension method*), 12
getConfig() (*gfm.tasklist.TaskListExtension method*), 16
getConfig() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19
getConfig() (*mdx_partial_gfm.PartialGithubFlavoredMarkdownExtens method*), 20
getConfigInfo() (*gfm.autolink.AutolinkExtension method*), 8
getConfigInfo() (*gfm.automail.AutomailExtension method*), 9
getConfigInfo() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
getConfigInfo() (*gfm.strikethrough.StrikethroughExtension method*), 13
getConfigInfo() (*gfm.tasklist.TaskListExtension method*), 16
getConfigInfo() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19

getConfigInfo() (*mdx_partial_gfm.PartialGithubFlavoredMarkdownExtension method*), 20
getConfigs() (*gfm.autolink.AutolinkExtension method*), 8
getConfigs() (*gfm.automail.AutomailExtension method*), 9
getConfigs() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
getConfigs() (*gfm.strikethrough.StrikethroughExtension method*), 13
getConfigs() (*gfm.tasklist.TaskListExtension method*), 16
getConfigs() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19
getConfigs() (*mdx_partial_gfm.PartialGithubFlavoredMarkdownExtension method*), 20
gfm (*module*), 17
gfm.autolink (*module*), 7
gfm.automail (*module*), 8
gfm.semi_sane_lists (*module*), 9
gfm.strikethrough (*module*), 12
gfm.tasklist (*module*), 13
GithubFlavoredMarkdownExtension (*class in mdx_gfm*), 18

H

handleMatch() (*gfm.autolink.AutolinkPattern method*), 8
handleMatch() (*gfm.automail.AutomailPattern method*), 9

L

lastChild() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 10
lastChild() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 11
looseDetab() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 11
looseDetab() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 11

M

mdx_gfm (*module*), 18
mdx_partial_gfm (*module*), 19

P

PartialGithubFlavoredMarkdownExtension (*class in mdx_partial_gfm*), 19

R

run() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 11
run() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 11
run() (*gfm.tasklist.TaskListProcessor method*), 16

S

SemiSaneListExtension (*class in gfm*), 17
SemiSaneListExtension (*class in gfm.semi_sane_lists*), 10
SemiSaneOListProcessor (*class in gfm.semi_sane_lists*), 10
SemiSaneUListProcessor (*class in gfm.semi_sane_lists*), 11
setConfig() (*gfm.autolink.AutolinkExtension method*), 8
setConfig() (*gfm.automail.AutomailExtension method*), 9
setConfig() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
setConfig() (*gfm.strikethrough.StrikethroughExtension method*), 13
setConfig() (*gfm.tasklist.TaskListExtension method*), 16
setConfig() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19
setConfig() (*mdx_partial_gfm.PartialGithubFlavoredMarkdownExtens method*), 20
setConfigs() (*gfm.autolink.AutolinkExtension method*), 8
setConfigs() (*gfm.automail.AutomailExtension method*), 9
setConfigs() (*gfm.semi_sane_lists.SemiSaneListExtension method*), 10
setConfigs() (*gfm.strikethrough.StrikethroughExtension method*), 13
setConfigs() (*gfm.tasklist.TaskListExtension method*), 16
setConfigs() (*mdx_gfm.GithubFlavoredMarkdownExtension method*), 19
setConfigs() (*mdx_partial_gfm.PartialGithubFlavoredMarkdownExte method*), 20
StandaloneFencedCodeExtension (*class in gfm*), 18
StrikethroughExtension (*class in gfm*), 18
StrikethroughExtension (*class in gfm.strikethrough*), 12

T

TaskListExtension (*class in gfm*), 18
TaskListExtension (*class in gfm.tasklist*), 15
TaskListProcessor (*class in gfm.tasklist*), 16
test() (*gfm.semi_sane_lists.SemiSaneOListProcessor method*), 11
test() (*gfm.semi_sane_lists.SemiSaneUListProcessor method*), 12
type() (*gfm.autolink.AutolinkPattern method*), 8
type() (*gfm.automail.AutomailPattern method*), 9

U

unescape() (*gfm.autolink.AutolinkPattern method*), 8
unescape() (*gfm.automail.AutomailPattern method*), 9