

---

# **Recommonmark Documentation**

*Release 0.7.1*

**Lu Zero, Eric Holscher, and contributors**

**May 18, 2021**



---

## Contents

---

<b>1</b>	<b>Contents</b>	<b>3</b>
<b>2</b>	<b>Getting Started</b>	<b>11</b>
<b>3</b>	<b>Development</b>	<b>15</b>
<b>4</b>	<b>Why a bridge?</b>	<b>17</b>
<b>5</b>	<b>Why another bridge to docutils?</b>	<b>19</b>
<b>6</b>	<b>Acknowledgement</b>	<b>21</b>
	<b>Index</b>	<b>23</b>



**Warning:** recommonmark is now deprecated. We recommend using [MyST](#) for a docutils bridge going forward. See [this issue](#) for background and discussion.

A `docutils`-compatibility bridge to [CommonMark](#).

This allows you to write CommonMark inside of Docutils & Sphinx projects.

Documentation is available on Read the Docs: <http://recommonmark.readthedocs.org>



## 1.1 API Reference

This document is for developers of recommonmark, it contains the API functions

### 1.1.1 Parser Component

**class** `recommonmark.parser.CommonMarkParser`

Bases: `docutils.parsers.Parser`

Docutils parser for CommonMark

**default\_depart** (*mdnode*)

Default node depart handler

If there is a matching `visit_<type>` method for a container node, then we should make sure to back up to it's parent element when the node is exited.

**depart\_heading** ( \_ )

Finish establishing section

Wrap up title node, but stick in the section node. Add the section names based on all the text nodes added to the title.

**parse** (*inputstring*, *document*)

Override to parse *inputstring* into document tree *document*.

### 1.1.2 Dummy State Machine

**class** `recommonmark.states.DummyStateMachine`

Bases: `docutils.statemachine.StateMachineWS`

A dummy state machine that mimicks the property of statemachine.

This state machine cannot be used for parsing, it is only used to generate directive and roles. Usage: - Call *reset* to reset the state - Then call *run\_directive* or *run\_role* to generate the node.

**get\_source\_and\_line** (*lineno=None*)

Return (source, line) tuple for current or given line number.

Looks up the source and line number in the *self.input\_lines* StringList instance to count for included source files.

If the optional argument *lineno* is given, convert it from an absolute line number to the corresponding (source, line) pair.

**reset** (*document, parent, level*)

Reset the state of state machine.

After reset, *self* and *self.state* can be used to passed to *docutils.parsers.rst.Directive.run*

### Parameters

- **document** (*docutils document*) – Current document of the node.
- **parent** (*parent node*) – Parent node that will be used to interpret role and directives.
- **level** (*int*) – Current section level.

**run\_directive** (*name, arguments=None, options=None, content=None*)

Generate directive node given arguments.

### Parameters

- **name** (*str*) – name of directive.
- **arguments** (*list*) – list of positional arguments.
- **options** (*dict*) – key value arguments.
- **content** (*content*) – content of the directive

**Returns node** – Node generated by the arguments.

**Return type** docutil Node

**run\_role** (*name, options=None, content=None*)

Generate a role node.

**options** [dict] key value arguments.

**content** [content] content of the directive

**Returns node** – Node generated by the arguments.

**Return type** docutil Node

### 1.1.3 AutoStructify Transformer

**class** `recommonmark.transform.AutoStructify` (*\*args, \*\*kwargs*)

Bases: `docutils.transforms.Transform`

Automatically try to transform blocks to sphinx directives.

This class is designed to handle AST generated by CommonMarkParser.

**apply** ()

Apply the transformation by configuration.



**auto\_code\_block** (*node*)

Try to automatically generate nodes for codeblock syntax.

**Parameters** **node** (*nodes.literal\_block*) – Original codeblock node

**Returns** **tocnode** – The converted toc tree node, None if conversion is not possible.

**Return type** docutils node

**auto\_inline\_code** (*node*)

Try to automatically generate nodes for inline literals.

**Parameters** **node** (*nodes.literal*) – Original codeblock node

**Returns** **tocnode** – The converted toc tree node, None if conversion is not possible.

**Return type** docutils node

**auto\_toc\_tree** (*node*)

Try to convert a list block to toctree in rst.

This function detects if the matches the condition and return a converted toc tree node. The matching condition: The list only contains one level, and only contains references

**Parameters** **node** (*nodes.Sequential*) – A list node in the doctree

**Returns** **tocnode** – The converted toc tree node, None if conversion is not possible.

**Return type** docutils node

**find\_replace** (*node*)

Try to find replace node for current node.

**Parameters** **node** (*docutil node*) – Node to find replacement for.

**Returns** **nodes** – The replacement nodes of current node. Returns None if no replacement can be found.

**Return type** node or list of node

**parse\_ref** (*ref*)

Analyze the ref block, and return the information needed.

**Parameters** **ref** (*nodes.reference*) –

**Returns** **result** – The returned result is tuple of (title, uri, docpath). title is the display title of the ref. uri is the html uri of to the ref after resolve. docpath is the absolute document path to the document, if the target corresponds to an internal document, this can be None

**Return type** tuple of (str, str, str)

**traverse** (*node*)

Traverse the document tree rooted at node.

**node** [docutil node] current root node to traverse

## 1.2 AutoStructify Component

AutoStructify is a component in recommonmark that takes a parsed docutil AST by `CommonMarkParser`, and transform it to another AST that introduces some of more. This enables additional features of recommonmark syntax, to introduce more structure into the final generated document.

## 1.2.1 Configuring AutoStructify

The behavior of AutoStructify can be configured via a dict in document setting. In sphinx, you can configure it by `conf.py`. The following snippet is what is actually used to generate this document, see full code at [conf.py](#).

```
github_doc_root = 'https://github.com/rtfd/recommonmark/tree/master/doc/'
def setup(app):
    app.add_config_value('recommonmark_config', {
        'url_resolver': lambda url: github_doc_root + url,
        'auto_toc_tree_section': 'Contents',
    }, True)
    app.add_transform(AutoStructify)
```

All the features are by default enabled

### List of options

- **enable\_auto\_toc\_tree**: whether enable *Auto Toc Tree* feature.
- **auto\_toc\_tree\_section**: when enabled, *Auto Toc Tree* will only be enabled on section that matches the title.
- **enable\_auto\_doc\_ref**: whether enable *Auto Doc Ref* feature. **Deprecated**
- **enable\_math**: whether enable *Math Formula*
- **enable\_inline\_math**: whether enable *Inline Math*
- **enable\_eval\_rst**: whether *Embed reStructuredText* is enabled.
- **url\_resolver**: a function that maps a existing relative position in the document to a http link

## 1.2.2 Auto Toc Tree

One of important command in tools like sphinx is `toctree`. This is a command to generate table of contents and tell sphinx about the structure of the documents. In markdown, usually we manually list of contents by a bullet list of url reference to the other documents.

AutoStructify transforms bullet list of document URLs like this

```
* [Title1] (doc1.md)
* [Title2] (doc2.md)
```

to the AST of this following reStructuredText code

```
.. toctree::
   :maxdepth: 1

   doc1
   doc2
```

You can also find the usage of this feature in `index.md` of this document.

## 1.2.3 Auto Doc Ref

---

**Note:** **This option is deprecated.** This option has been superseded by the default linking behavior, which will first try to resolve as an internal reference, and then as an external reference.

---

It is common to refer to another document page in one document. We usually use reference to do that. AutoStructify will translate these reference block into a structured document reference. For example

```
[API Reference] (api_ref.md)
```

will be translated to the AST of following reStructuredText code

```
:doc:`API Reference </api_ref>`
```

And it will be rendered as *API Reference*

## 1.2.4 URL Resolver

Sometimes in a markdown, we want to refer to the code in the same repo. This can usually be done by a reference by reference path. However, since the generated document is hosted elsewhere, the relative path may not work in generated document site. URL resolver is introduced to solve this problem.

Basically, you can define a function that maps an relative path of document to the http path that you wish to link to. For example, the setting mentioned in the beginning of this document used a resolver that maps the files to github. So `[parser code] (../recommonmark/parser.py)` will be translated into [parser code](#)

Note that the reference to the internal document will not be passed to url resolver, and will be linked to the internal document pages correctly, see *Auto Doc Ref*.

## 1.2.5 Codeblock Extensions

In markdown, you can write codeblocks fenced by (at least) three backticks (````). The following is an example of codeblock.

```
``` language
some code block
```
```

Codeblock extensions are mechanism that specialize certain codeblocks to different render behaviors. The extension will be trigger by the language argument to the codeblock

### Syntax Highlight

You can highlight syntax of codeblocks by specifying the language you need. For example,

```
```python
def function():
    return True
```
```

will be rendered as

```
def function():
    return True
```

### Math Formula

You can normally write latex math formula with `math` codeblock. See also *Inline Math*.

**Warning:** Math is currently broken on some Sphinx builds.

Example

```
```math
E = m c^2
```
```

will be rendered as

```
E = m c^2
```

### Embed reStructuredText

Recommonmark also allows embedding reStructuredText syntax in the codeblocks. There are two styles for embedding reStructuredText. The first is enabled by `eval_rst` codeblock. The content of codeblock will be parsed as reStructuredText and insert into the document. This can be used to quickly introduce some of reStructuredText command that not yet available in markdown. For example,

```
```eval_rst
.. autoclass:: recommonmark.transform.AutoStructify
   :show-inheritance:
```
```

will be rendered as

```
class recommonmark.transform.AutoStructify (*args, **kwargs)
    Bases: docutils.transforms.Transform

    Automatically try to transform blocks to sphinx directives.

    This class is designed to handle AST generated by CommonMarkParser.
```

This example used to use sphinx autodoc to insert document of AutoStructify class definition into the document.

The second style is a shorthand of the above style. It allows you to leave off the `eval_rst ..` portion and directly render directives. For example,

```
``` important:: Its a note! in markdown!
```
```

will be rendered as

---

**Important:** Its a note! in markdown!

---

### An Advanced Example

```
``` sidebar:: Line numbers and highlights

   emphasis-lines:
     highlights the lines.
```

(continues on next page)

(continued from previous page)

```

linenos:
  shows the line numbers as well.
caption:
  shown at the top of the code block.
name:
  may be referenced with `:ref:` later.
...

```code-block:: markdown
:linenos:
:emphasize-lines: 3,5
:caption: An example code-block with everything turned on.
:name: Full code-block example

# Comment line
import System
System.run_emphasis_line
# Long lines in code blocks create a auto horizontal scrollbar
System.exit!
...

```

will be rendered as

### Line numbers and highlights

**emphasize-lines:** highlights the lines.

**linenos:** shows the line numbers as well.

**caption:** shown at the top of the code block.

**name:** may be referenced with `:ref:` later.

Listing 1: An example code-block with everything turned on.

```

1 # Comment line
2 import System
3 System.run_emphasis_line
4 # Long lines in code blocks create a auto horizontal scrollbar
5 System.exit!

```

The `<div style="clear: right;"></div>` line clears the sidebar for the next title.

## 1.2.6 Inline Math

Besides the *Math Formula*, you can also write latex math in inline codeblock text. You can do so by inserting `$` in the beginning and end of inline codeblock.

Example

**Warning:** Math is currently broken on some Sphinx builds.

```
This formula ``$ y=\sum_{i=1}^n g(x_i) $`
```

will be rendered as:

This formula  $y = \sum_{i=1}^n g(x_i)$

To use `recommonmark` inside of Sphinx only takes 2 steps. First you install it:

```
pip install recommonmark
```

Then add this to your Sphinx `conf.py`:

```
# for Sphinx-1.4 or newer
extensions = ['recommonmark']

# for Sphinx-1.3
from recommonmark.parser import CommonMarkParser

source_parsers = {
    '.md': CommonMarkParser,
}

source_suffix = ['.rst', '.md']
```

This allows you to write both `.md` and `.rst` files inside of the same project.

## 2.1 Links

For all links in commonmark that aren't explicit URLs, they are treated as cross references with the `:any:` role. This allows referencing a lot of things including files, labels, and even objects in the loaded domain.

### 2.1.1 Linking to headings in other files

For linking to headings in other files you can use the `autosectionlabel` sphinx feature, e.g.

```
# conf.py

extensions = [
    # Auto-generate section labels.
    'sphinx.ext.autosectionlabel',
]

# Prefix document path to section labels, otherwise autogenerated labels would look
# ↳ like 'heading'
# rather than 'path/to/file:heading'
autosectionlabel_prefix_document = True
```

You would use it like:

```
<!-- path/to/file_1.md -->

# Title

## My Subtitle
```

```
<!-- file_2.md -->

[My Subtitle] []

[My Subtitle]: <path/to/file_1:My Subtitle>
```

## 2.2 AutoStructify

AutoStructify makes it possible to write your documentation in Markdown, and automatically convert this into rST at build time. See the [AutoStructify Documentation](#) for more information about configuration and usage.

To use the advanced markdown to rst transformations you must add `AutoStructify` to your Sphinx `conf.py`.

```
# At top on conf.py (with other import statements)
import recommonmark
from recommonmark.transform import AutoStructify

# At the bottom of conf.py
def setup(app):
    app.add_config_value('recommonmark_config', {
        'url_resolver': lambda url: github_doc_root + url,
        'auto_toc_tree_section': 'Contents',
    }, True)
    app.add_transform(AutoStructify)
```

See <https://github.com/rtfd/recommonmark/blob/master/docs/conf.py> for a full example.

AutoStructify comes with the following options. See [http://recommonmark.readthedocs.org/en/latest/auto\\_structify.html](http://recommonmark.readthedocs.org/en/latest/auto_structify.html) for more information about the specific features.

- **enable\_auto\_toc\_tree**: enable the Auto Toc Tree feature.
- **auto\_toc\_maxdepth**: The max depth of the Auto Toc. Defaults to 1.
- **auto\_toc\_tree\_section**: when True, Auto Toc Tree will only be enabled on section that matches the title.
- **enable\_auto\_doc\_ref**: enable the Auto Doc Ref feature. **Deprecated**



- **enable\_math**: enable the Math Formula feature.
- **enable\_inline\_math**: enable the Inline Math feature.
- **enable\_eval\_rst**: enable the evaluate embedded reStructuredText feature.
- **url\_resolver**: a function that maps a existing relative position in the document to a http link
- **known\_url\_schemes**: a list of url schemes to treat as URLs, schemes not in this list will be assumed to be Sphinx cross-references. Defaults to `None`, which means treat all URL schemes as URLs. Example: `['http', 'https', 'mailto']`



## CHAPTER 3

---

### Development

---

You can run the tests by running `tox` in the top-level of the project.

We are working to expand test coverage, but this will at least test basic Python 2 and 3 compatability.



## CHAPTER 4

---

### Why a bridge?

---

Many python tools (mostly for documentation creation) rely on `docutils`. But `docutils` only supports a `ReStructuredText` syntax.

For instance [this issue](#) and [this StackOverflow question](#) show that there is an interest in allowing `docutils` to use markdown as an alternative syntax.



---

### Why another bridge to docutils?

---

`recommonmark` uses the `python` implementation of `CommonMark` while `remarkdown` implements a stand-alone parser leveraging `parsley`.

Both output a `docutils` document tree and provide scripts that leverage `docutils` for generation of different types of documents.





## CHAPTER 6

---

### Acknowledgement

---

recommonmark is mainly derived from [remarkdown](#) by Steve Genoud and leverages the python CommonMark implementation.

It was originally created by [Luca Barbato](#), and is now maintained in the Read the Docs (rtd) GitHub organization.



**A**

`apply()` (*recommonmark.transform.AutoStructify method*), 4  
`auto_code_block()` (*recommonmark.transform.AutoStructify method*), 4  
`auto_inline_code()` (*recommonmark.transform.AutoStructify method*), 5  
`auto_toc_tree()` (*recommonmark.transform.AutoStructify method*), 5  
`AutoStructify` (*class in recommonmark.transform*), 4, 8

**C**

`CommonMarkParser` (*class in recommonmark.parser*), 3

**D**

`default_depart()` (*recommonmark.parser.CommonMarkParser method*), 3  
`depart_heading()` (*recommonmark.parser.CommonMarkParser method*), 3  
`DummyStateMachine` (*class in recommonmark.states*), 3

**F**

`find_replace()` (*recommonmark.transform.AutoStructify method*), 5

**G**

`get_source_and_line()` (*recommonmark.states.DummyStateMachine method*), 4

**P**

`parse()` (*recommonmark.parser.CommonMarkParser method*), 3

`parse_ref()` (*recommonmark.transform.AutoStructify method*), 5

**R**

`reset()` (*recommonmark.states.DummyStateMachine method*), 4  
`run_directive()` (*recommonmark.states.DummyStateMachine method*), 4  
`run_role()` (*recommonmark.states.DummyStateMachine method*), 4

**T**

`traverse()` (*recommonmark.transform.AutoStructify method*), 5