

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.5.5
January 8, 2018

Contents

1	Introduction	1	2.3	W _T _E X Interface	26
1.1	About Markdown	1	2.4	ConT _E Xt Interface	35
1.2	Feedback	2			
1.3	Acknowledgements	2	3	Technical Documentation	36
1.4	Prerequisites	2	3.1	Lua Implementation	36
2	User Guide	5	3.2	Plain T _E X Implementation	83
2.1	Lua Interface	5	3.3	W _T _E X Implementation	92
2.2	Plain T _E X Interface	12	3.4	ConT _E Xt Implementation	98

1 Introduction

This document is a reference manual for the Markdown package. It is split into three sections. This section explains the purpose and the background of the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package. Section 3 describes the implementation of the package. It is aimed at the developer of the package and the curious user.

1.1 About Markdown

The Markdown package provides facilities for the conversion of markdown markup to plain T_EX. These are provided both in the form of a Lua module and in the form of plain T_EX, W_T_EX, and ConT_EXt macro packages that enable the direct inclusion of markdown documents inside T_EX documents.

Architecturally, the package consists of the Lunamark v0.5.0 Lua module by John MacFarlane, which was slimmed down and rewritten for the needs of the package. On top of Lunamark sits code for the plain T_EX, W_T_EX, and ConT_EXt formats by Vít Novotný.

```
1 local metadata = {
2   version   = "2.5.5",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Novotný",
5   copyright = "2009-2016 John MacFarlane, Hans Hagen; " ..
```

```

6             "2016-2018 Vít Novotný",
7     license   = "LPPL 1.3"
8 }
9 if not modules then modules = { } end
10 modules['markdown'] = metadata

```

1.2 Feedback

Please use the markdown project page on GitHub¹ to report bugs and submit feature requests. Before making a feature request, please ensure that you have thoroughly studied this manual. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question on the T_EX-~~E~~_TX Stack Exchange².

1.3 Acknowledgements

I would like to thank the Faculty of Informatics at the Masaryk University in Brno for providing me with the opportunity to work on this package alongside my studies. I would also like to thank the creator of the Lunamark Lua module, John Macfarlane, for releasing Lunamark under a permissive license that enabled its inclusion into the package.

The T_EX part of the package draws inspiration from several sources including the source code of ~~E~~T_EX 2_ε, the minted package by Geoffrey M. Poore – which likewise tackles the issue of interfacing with an external interpreter from T_EX, the filecontents package by Scott Pakin, and others.

1.4 Prerequisites

This section gives an overview of all resources required by the package.

1.4.1 Lua Prerequisites

The Lua part of the package requires that the following Lua modules are available from within the LuaT_EX engine:

LPeg \geq 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg \geq 0.10 is included in LuaT_EX \geq 0.72.0 (T_EXLive \geq 2013).

```

11     local lpeg = require("lpeg")

```

¹<https://github.com/witiko/markdown/issues>

²<https://tex.stackexchange.com>

Selene Unicode A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and footnote tags to the lower case. Selene Unicode is included in all releases of LuaT_EX (T_EXLive \geq 2008).

```
12      local unicode = require("unicode")
```

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaT_EX (T_EXLive \geq 2008).

```
13      local md5 = require("md5")
```

All the abovelisted modules are statically linked into the current version of the LuaT_EX engine (see [1, Section 3.3]).

1.4.2 Plain T_EX Prerequisites

The plain T_EX part of the package requires that the plain T_EX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.4.1), and the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain T_EX code to create the cache directory specified by the `\markdownOptionCacheDir` macro before interfacing with the Lunamark library. Lua File System is included in all releases of LuaT_EX (T_EXLive \geq 2008).

The plain T_EX code makes use of the `isdir` method that was added to the Lua File System library by the LuaT_EX engine developers (see [1, Section 3.2]).

The Lua File System module is statically linked into the LuaT_EX engine (see [1, Section 3.3]).

The plain T_EX part of the package also requires that either the LuaT_EX `\directlua` primitive or the shell access file stream 18 is available in your T_EX engine. If only the shell access file stream is available in your T_EX engine (as is the case with pdfT_EX and X_ET_EX) or if you enforce the use of shell using the `\markdownMode` macro, then note that unless your T_EX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.4.3 L^AT_EX Prerequisites

The L^AT_EX part of the package requires that the L^AT_EX 2_ε format is loaded,

```
14 \NeedsTeXFormat{LaTeX2e}%
```

all the plain T_EX prerequisites (see Section 1.4.2), and the following L^AT_EX 2_ε packages:

keyval A package that enables the creation of parameter sets. This package is used to provide the `\markdownSetup` macro, the package options processing, as well as the parameters of the `markdown*` \TeX environment.

15 `\RequirePackage{keyval}`

url A package that provides the `\url` macro for the typesetting of URLs. It is used to provide the default token renderer prototype (see Section 2.2.4) for links.

16 `\RequirePackage{url}`

graphicx A package that provides the `\includegraphics` macro for the typesetting of images. It is used to provide the corresponding default token renderer prototype (see Section 2.2.4).

17 `\RequirePackage{graphicx}`

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists. It is used to provide the corresponding default token renderer prototypes (see Section 2.2.4).

ifthen A package that provides a concise syntax for the inspection of macro values. It is used to determine whether or not the `paralist` package should be loaded based on the user options.

18 `\RequirePackage{ifthen}`

fancyvrb A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code. It is used to provide the corresponding default token renderer prototype (see Section 2.2.4).

19 `\RequirePackage{fancyvrb}`

csvsimple A package that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.4).

20 `\RequirePackage{csvsimple}`

1.4.4 Con \TeX t prerequisites

The Con \TeX t part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain \TeX prerequisites (see Section 1.4.2), and the following Con \TeX t modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.4).

2 User Guide

This part of the manual describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither \TeX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is purely abstract. It serves as a means of structuring this manual and as a promise to the user that if they only access the package through the interfaces, the future versions of the package should remain backwards compatible.

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain \TeX . This interface is used by the plain \TeX implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

The Lua interface is implemented by the `markdown` Lua module.

```
21 local M = {}
```

2.1.1 Conversion from Markdown to Plain \TeX

The Lua interface exposes the `new(options)` method. This method creates converter functions that perform the conversion from markdown to plain \TeX according to the table `options` that contains options recognized by the Lua interface. (see Section 2.1.2). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `_Hello world!_` to a \TeX output using the default options and prints the \TeX output:

```
local md = require("markdown")
local convert = md.new()
print(convert("_Hello world!_"))
```

2.1.2 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` table.

```
22 local defaultOptions = {}
```

2.1.2.1 File and Directory Names

`cacheDir`= $\langle path \rangle$ default: .

A path to the directory containing auxiliary cache files. It is considered sane to set the path to a subdirectory of `outputDir`. If the last segment of the path does not exist, it will be created by the plain \TeX , \LaTeX , and Con \TeX t implementations. The Lua implementation expects that the entire path already exists.

When iteratively writing and typesetting a markdown document, the cache files are going to accumulate over time. You are advised to clean the cache directory every now and then, or to set it to a temporary filesystem (such as `/tmp` on UN*X systems), which gets periodically emptied.

```
23 defaultOptions.cacheDir = "."
```

`outputDir`= $\langle path \rangle$ default: .

A path to the directory containing output files. The path must be set to the same value as the `-output-directory` option of your \TeX engine for the package to function correctly.

```
24 defaultOptions.outputDir = "."
```

2.1.2.2 Parser Options

`blankBeforeBlockquote`=true, false default: false

<code>true</code>	Require a blank line between a paragraph and the following blockquote.
<code>false</code>	Do not require a blank line between a paragraph and the following blockquote.

```
25 defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence`=true, false default: false

<code>true</code>	Require a blank line between a paragraph and the following fenced code block.
<code>false</code>	Do not require a blank line between a paragraph and the following fenced code block.

```
26 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeHeading=true, false` default: false

- `true` Require a blank line between a paragraph and the following header.
- `false` Do not require a blank line between a paragraph and the following header.

27 `defaultOptions.blankBeforeHeading = false`

`breakableBlockquotes=true, false` default: false

- `true` A blank line separates block quotes.
- `false` Blank lines in the middle of a block quote are ignored.

28 `defaultOptions.breakableBlockquotes = false`

`citationNbsps=true, false` default: false

- `true` Replace regular spaces with non-breakable spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.
- `false` Do not replace regular spaces with non-breakable spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

29 `defaultOptions.citationNbsps = true`

`citations=true, false` default: false

- `true` Enable the pandoc citation syntax extension:

Here is a simple parenthetical citation [doe99] and here is a string of several [see doe99, pp. 33-35; also smith04, chap. 1].

A parenthetical citation can have a [prenote doe99] and a [smith04 postnote]. The name of the author can be suppressed by inserting a dash before the name of an author as follows [-smith04].

Here is a simple text citation doe99 and here is a string of several doe99 [pp. 33-35; also smith04, chap. 1]. Here is one with the name of the author suppressed -doe99.

`false` Disable the pandoc citation syntax extension.

```
30 defaultOptions.citations = false
```

`codeSpans=true, false`

default: true

`true` Enable the code span syntax:

```
Use the 'printf()' function.
''There is a literal backtick (') here.''
```

`false` Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

```
''This is a quote.''
```

```
31 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

`true` Enable the iA Writer content blocks syntax extension [2]:

```
http://example.com/minard.jpg (Napoleon's disastrous
Russian campaign of 1812)
/Flowchart.png "Engineering Flowchart"
/Savings Account.csv 'Recent Transactions'
/Example.swift
/Lorem Ipsum.txt
```

`false` Disable the iA Writer content blocks syntax extension.

```
32 defaultOptions.contentBlocks = false
```

`contentBlocksLanguageMap=<filename>`

default: markdown-languages.json

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks. See Section 2.2.3.9 for more information.

```
33 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```


`definitionLists=true, false`

default: false

`true`

Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

Third paragraph of definition 2.
```

`false`

Disable the pandoc definition list syntax extension.

```
34 defaultOptions.definitionLists = false
```

`fencedCode=true, false`

default: false

`true`

Enable the commonmark fenced code block extension:

```
~~~ js
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
~~~~~

''' html
<pre>
  <code>
    // Some comments
    line 1 of code
    line 2 of code
    line 3 of code
  </code>
</pre>
'''
```

`false`

Disable the commonmark fenced code block extension.

```
35 defaultOptions.fencedCode = false
```

`footnotes=true, false`

default: false

`true`

Enable the pandoc footnote syntax extension:

```
Here is a footnote reference, [^1] and another. [^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they
    belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the
    first line. In this way, multi-paragraph footnotes
    work like multi-paragraph list items.

This paragraph won't be part of the note, because it
isn't indented.
```

`false`

Disable the pandoc footnote syntax extension.

```
36 defaultOptions.footnotes = false
```

`hashEnumerators=true, false`

default: false

`true`

Enable the use of hash symbols (#) as ordered item list markers:

```
#. Bird
#. McHale
#. Parish
```

`false`

Disable the use of hash symbols (#) as ordered item list markers.

```
37 defaultOptions.hashEnumerators = false
```

`html=true, false`

default: false

`true`

Enable the recognition of HTML tags, block elements, comments, HTML instructions, and entities in the input. Tags, block elements (along with contents), HTML instructions, and comments will be ignored and HTML entities will be replaced with the corresponding Unicode codepoints.

`false` Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```
38 defaultOptions.html = false
```

`hybrid=true, false` default: false

`true` Disable the escaping of special plain \TeX characters, which makes it possible to intersperse your markdown markup with \TeX code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix \TeX and markdown markup freely.

`false` Enable the escaping of special plain \TeX characters outside verbatim environments, so that they are not interpreted by \TeX . This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
39 defaultOptions.hybrid = false
```

`inlineFootnotes=true, false` default: false

`true` Enable the pandoc inline footnote syntax extension:

Here is an inline note.^[Inlines notes are easier to write, since you don't have to pick an identifier and move down to type the note.]

`false` Disable the pandoc inline footnote syntax extension.

```
40 defaultOptions.inlineFootnotes = false
```

`preserveTabs=true, false` default: false

`true` Preserve all tabs in the input.

`false` Convert any tabs in the input to spaces.

```
41 defaultOptions.preserveTabs = false
```

`smartEllipses=true, false` default: false

`true` Convert any ellipses in the input to the `\markdownRendererEllipsis` \TeX macro.

`false` Preserve all ellipses in the input.

```
42 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

true Make the number in the first item in ordered lists significant. The item numbers will be passed to the `\markdownRendererOlItemWithNumber` \TeX macro.

false Ignore the number in the items of ordered lists. Each item will only produce a `\markdownRendererOlItem` \TeX macro.

```
43 defaultOptions.startNumber = true
```

`tightLists=true, false` default: true

true Lists whose bullets do not consist of multiple paragraphs will be detected and passed to the `\markdownRendererOlBeginTight`, `\markdownRendererOlEndTight`, `\markdownRendererUlBeginTight`, `\markdownRendererUlEndTight`, `\markdownRendererDlBeginTight`, and `\markdownRendererDlEndTight` macros.

false Lists whose bullets do not consist of multiple paragraphs will be treated the same way as lists that do.

```
44 defaultOptions.tightLists = true
```

`underscores=true, false` default: true

true Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```
*single asterisks*
_single underscores_
**double asterisks**
__double underscores__
```

false Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the `hybrid` option without the need to constantly escape subscripts.

```
45 defaultOptions.underscores = true
```

2.2 Plain \TeX Interface

The plain \TeX interface provides macros for the typesetting of markdown input from within plain \TeX , for setting the Lua interface options (see Section 2.1.2) used during

the conversion from markdown to plain T_EX, and for changing the way markdown the tokens are rendered.

```
46 \def\markdownLastModified{2018/01/08}%  
47 \def\markdownVersion{2.5.5}%
```

The plain T_EX interface is implemented by the `markdown.tex` file that can be loaded as follows:

```
\input markdown
```

It is expected that the special plain T_EX characters have the expected category codes, when `\input`ting the file.

2.2.1 Typesetting Markdown

The interface exposes the `\markdownBegin`, `\markdownEnd`, and `\markdownInput` macros.

The `\markdownBegin` macro marks the beginning of a markdown document fragment and the `\markdownEnd` macro marks its end.

```
48 \let\markdownBegin\relax  
49 \let\markdownEnd\relax
```

You may prepend your own code to the `\markdownBegin` macro and redefine the `\markdownEnd` macro to produce special effects before and after the markdown block.

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T_EX (see [3, p. 46]). As a corollary, the `\markdownBegin` macro also ignores them.

The `\markdownBegin` and `\markdownEnd` macros will also consume the rest of the lines at which they appear. In the following example plain T_EX code, the characters `c`, `e`, and `f` will not appear in the output.

```
\input markdown  
a  
b \markdownBegin c  
d  
e \markdownEnd f  
g  
\bye
```

Note that you may also not nest the `\markdownBegin` and `\markdownEnd` macros. The following example plain \TeX code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

```
\input markdown
\markdownBegin
_Hello_ **world** ...
\markdownEnd
\bye
```

The `\markdownInput` macro accepts a single parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain \TeX .

```
50 \let\markdownInput\relax
```

This macro is not subject to the abovelisted limitations of the `\markdownBegin` and `\markdownEnd` macros.

The following example plain \TeX code showcases the usage of the `\markdownInput` macro:

```
\input markdown
\markdownInput{hello.md}
\bye
```

2.2.2 Options

The plain \TeX options are represented by \TeX macros. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.2), while some of them are specific to the plain \TeX interface.

2.2.2.1 File and Directory names The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain \TeX in \TeX engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that \TeX engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
51 \def\markdownOptionHelperScriptFileName{\jobname.markdown.lua}%
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the conversion from markdown to plain \TeX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same

limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```
52 \def\markdownOptionInputTempFileName{\jobname.markdown.in}%
```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain \TeX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
53 \def\markdownOptionOutputTempFileName{\jobname.markdown.out}%
```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain \TeX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
54 \def\markdownOptionErrorTempFileName{\jobname.markdown.err}%
```

The `\markdownOptionCacheDir` macro corresponds to the Lua interface `cacheDir` option that sets the name of the directory that will contain the produced cache files. The option defaults to `_markdown_\jobname`, which is a similar naming scheme to the one used by the minted \LaTeX package. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
55 \def\markdownOptionCacheDir{\markdownOptionOutputDir/_markdown_\jobname}%
```

The `\markdownOptionOutputDir` macro corresponds to the Lua interface `outputDir` option that sets the name of the directory that will contain the produced cache files. The option defaults to `..`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
56 \def\markdownOptionOutputDir{.}%
```

2.2.2.2 Lua Interface Options The following macros map directly to the options recognized by the Lua interface (see Section 2.1.2) and are not processed by the plain \TeX implementation, only passed along to Lua. They are undefined, which makes them fall back to the default values provided by the Lua interface.

For the macros that correspond to the non-boolean options recognized by the Lua interface, the same limitations apply here in the case of the `\markdownOptionHelperScriptFileName` macro.

```
57 \let\markdownOptionBlankBeforeBlockquote\undefined
58 \let\markdownOptionBlankBeforeCodeFence\undefined
59 \let\markdownOptionBlankBeforeHeading\undefined
60 \let\markdownOptionBreakableBlockquotes\undefined
61 \let\markdownOptionCitations\undefined
62 \let\markdownOptionCitationNbsps\undefined
63 \let\markdownOptionContentBlocks\undefined
```

```

64 \let\markdownOptionContentBlocksLanguageMap\undefined
65 \let\markdownOptionDefinitionLists\undefined
66 \let\markdownOptionFootnotes\undefined
67 \let\markdownOptionFencedCode\undefined
68 \let\markdownOptionHashEnumerators\undefined
69 \let\markdownOptionHtml\undefined
70 \let\markdownOptionHybrid\undefined
71 \let\markdownOptionInlineFootnotes\undefined
72 \let\markdownOptionPreserveTabs\undefined
73 \let\markdownOptionSmartEllipses\undefined
74 \let\markdownOptionStartNumber\undefined
75 \let\markdownOptionTightLists\undefined

```

2.2.3 Token Renderers

The following TeX macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.4).

2.2.3.1 Interblock Separator Renderer The `\markdownRendererInterblockSeparator` macro represents a separator between two markdown block elements. The macro receives no arguments.

```

76 \def\markdownRendererInterblockSeparator{%
77   \markdownRendererInterblockSeparatorPrototype}%

```

2.2.3.2 Line Break Renderer The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

```

78 \def\markdownRendererLineBreak{%
79   \markdownRendererLineBreakPrototype}%

```

2.2.3.3 Ellipsis Renderer The `\markdownRendererEllipsis` macro replaces any occurrence of ASCII ellipses in the input text. This macro will only be produced, when the `smartEllipses` option is `true`. The macro receives no arguments.

```

80 \def\markdownRendererEllipsis{%
81   \markdownRendererEllipsisPrototype}%

```

2.2.3.4 Non-breaking Space Renderer The `\markdownRendererNbsp` macro represents a non-breaking space.

```

82 \def\markdownRendererNbsp{%
83   \markdownRendererNbspPrototype}%

```


2.2.3.5 Special Character Renderers The following macros replace any special plain \TeX characters (including the active pipe character (`|`) of Con \TeX t) in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```

84 \def\markdownRendererLeftBrace{%
85   \markdownRendererLeftBracePrototype}%
86 \def\markdownRendererRightBrace{%
87   \markdownRendererRightBracePrototype}%
88 \def\markdownRendererDollarSign{%
89   \markdownRendererDollarSignPrototype}%
90 \def\markdownRendererPercentSign{%
91   \markdownRendererPercentSignPrototype}%
92 \def\markdownRendererAmpersand{%
93   \markdownRendererAmpersandPrototype}%
94 \def\markdownRendererUnderscore{%
95   \markdownRendererUnderscorePrototype}%
96 \def\markdownRendererHash{%
97   \markdownRendererHashPrototype}%
98 \def\markdownRendererCircumflex{%
99   \markdownRendererCircumflexPrototype}%
100 \def\markdownRendererBackslash{%
101   \markdownRendererBackslashPrototype}%
102 \def\markdownRendererTilde{%
103   \markdownRendererTildePrototype}%
104 \def\markdownRendererPipe{%
105   \markdownRendererPipePrototype}%

```

2.2.3.6 Code Span Renderer The `\markdownRendererCodeSpan` macro represents inlined code span in the input text. It receives a single argument that corresponds to the inlined code span.

```

106 \def\markdownRendererCodeSpan{%
107   \markdownRendererCodeSpanPrototype}%

```

2.2.3.7 Link Renderer The `\markdownRendererLink` macro represents a hyper-link. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```

108 \def\markdownRendererLink{%
109   \markdownRendererLinkPrototype}%

```

2.2.3.8 Image Renderer The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```

110 \def\markdownRendererImage{%
111   \markdownRendererImagePrototype}%

```

2.2.3.9 Content Block Renderers The `\markdownRendererContentBlock` macro represents an iA Writer content block. It receives four arguments: the local file or online image filename extension cast to the lower case, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

```
112 \def\markdownRendererContentBlock{%
113   \markdownRendererContentBlockPrototype}%
```

The `\markdownRendererContentBlockOnlineImage` macro represents an iA Writer online image content block. The macro receives the same arguments as `\markdownRendererContentBlock`.

```
114 \def\markdownRendererContentBlockOnlineImage{%
115   \markdownRendererContentBlockOnlineImagePrototype}%
```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension s . If any `markdown-languages.json` file found by `kpathsea`³ contains a record (k, v) , then a non-online-image content block with the filename extension $s, s:\text{lower}() = k$ is considered to be in a known programming language v . The macro receives five arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local TeX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by [2] is a good starting point.

```
116 \def\markdownRendererContentBlockCode{%
117   \markdownRendererContentBlockCodePrototype}%
```

2.2.3.10 Bullet List Renderers The `\markdownRendererUlBegin` macro represents the beginning of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
118 \def\markdownRendererUlBegin{%
119   \markdownRendererUlBeginPrototype}%
```

The `\markdownRendererUlBeginTight` macro represents the beginning of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
120 \def\markdownRendererUlBeginTight{%
121   \markdownRendererUlBeginTightPrototype}%
```

³Local files take precedence. Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

The `\markdownRendererUListItem` macro represents an item in a bulleted list. The macro receives no arguments.

```
122 \def\markdownRendererUListItem{%  
123   \markdownRendererUListItemPrototype}%
```

The `\markdownRendererUListItemEnd` macro represents the end of an item in a bulleted list. The macro receives no arguments.

```
124 \def\markdownRendererUListItemEnd{%  
125   \markdownRendererUListItemEndPrototype}%
```

The `\markdownRendererUListEnd` macro represents the end of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
126 \def\markdownRendererUListEnd{%  
127   \markdownRendererUListEndPrototype}%
```

The `\markdownRendererUListEndTight` macro represents the end of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
128 \def\markdownRendererUListEndTight{%  
129   \markdownRendererUListEndTightPrototype}%
```

2.2.3.11 Ordered List Renderers The `\markdownRendererOlBegin` macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
130 \def\markdownRendererOlBegin{%  
131   \markdownRendererOlBeginPrototype}%
```

The `\markdownRendererOlBeginTight` macro represents the beginning of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
132 \def\markdownRendererOlBeginTight{%  
133   \markdownRendererOlBeginTightPrototype}%
```

The `\markdownRendererOlItem` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is `false`. The macro receives no arguments.

```
134 \def\markdownRendererOlItem{%  
135   \markdownRendererOlItemPrototype}%
```

The `\markdownRendererOlItemEnd` macro represents the end of an item in an ordered list. The macro receives no arguments.

```
136 \def\markdownRendererOlItemEnd{%  
137   \markdownRendererOlItemEndPrototype}%
```

The `\markdownRendererOlItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is `true`. The macro receives no arguments.

```
138 \def\markdownRendererOlItemWithNumber{%
139   \markdownRendererOlItemWithNumberPrototype}%
```

The `\markdownRendererOlEnd` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
140 \def\markdownRendererOlEnd{%
141   \markdownRendererOlEndPrototype}%
```

The `\markdownRendererOlEndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
142 \def\markdownRendererOlEndTight{%
143   \markdownRendererOlEndTightPrototype}%
```

2.2.3.12 Definition List Renderers The following macros are only produces, when the `definitionLists` option is `true`.

The `\markdownRendererDlBegin` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
144 \def\markdownRendererDlBegin{%
145   \markdownRendererDlBeginPrototype}%
```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```
146 \def\markdownRendererDlBeginTight{%
147   \markdownRendererDlBeginTightPrototype}%
```

The `\markdownRendererDlItem` macro represents a term in a definition list. The macro receives a single argument that corresponds to the term being defined.

```
148 \def\markdownRendererDlItem{%
149   \markdownRendererDlItemPrototype}%
```

The `\markdownRendererDlItemEnd` macro represents the end of a list of definitions for a single term.

```
150 \def\markdownRendererDlItemEnd{%
151   \markdownRendererDlItemEndPrototype}%
```

The `\markdownRendererDlDefinitionBegin` macro represents the beginning of a definition in a definition list. There can be several definitions for a single term.

```

152 \def\markdownRendererDlDefinitionBegin{%
153   \markdownRendererDlDefinitionBeginPrototype}%

```

The `\markdownRendererDlDefinitionEnd` macro represents the end of a definition in a definition list. There can be several definitions for a single term.

```

154 \def\markdownRendererDlDefinitionEnd{%
155   \markdownRendererDlDefinitionEndPrototype}%

```

The `\markdownRendererDlEnd` macro represents the end of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

156 \def\markdownRendererDlEnd{%
157   \markdownRendererDlEndPrototype}%

```

The `\markdownRendererDlEndTight` macro represents the end of a definition list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

158 \def\markdownRendererDlEndTight{%
159   \markdownRendererDlEndTightPrototype}%

```

2.2.3.13 Emphasis Renderers The `\markdownRendererEmphasis` macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

160 \def\markdownRendererEmphasis{%
161   \markdownRendererEmphasisPrototype}%

```

The `\markdownRendererStrongEmphasis` macro represents a strongly emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

162 \def\markdownRendererStrongEmphasis{%
163   \markdownRendererStrongEmphasisPrototype}%

```

2.2.3.14 Block Quote Renderers The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```

164 \def\markdownRendererBlockQuoteBegin{%
165   \markdownRendererBlockQuoteBeginPrototype}%

```

The `\markdownRendererBlockQuoteEnd` macro represents the end of a block quote. The macro receives no arguments.

```

166 \def\markdownRendererBlockQuoteEnd{%
167   \markdownRendererBlockQuoteEndPrototype}%

```

2.2.3.15 Code Block Renderers The `\markdownRendererInputVerbatim` macro represents a code block. The macro receives a single argument that corresponds to the filename of a file containing the code block contents.

```
168 \def\markdownRendererInputVerbatim{%  
169   \markdownRendererInputVerbatimPrototype}%
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is `true`. The macro receives two arguments that correspond to the filename of a file containing the code block contents and to the code fence infostring.

```
170 \def\markdownRendererInputFencedCode{%  
171   \markdownRendererInputFencedCodePrototype}%
```

2.2.3.16 Heading Renderers The `\markdownRendererHeadingOne` macro represents a first level heading. The macro receives a single argument that corresponds to the heading text.

```
172 \def\markdownRendererHeadingOne{%  
173   \markdownRendererHeadingOnePrototype}%
```

The `\markdownRendererHeadingTwo` macro represents a second level heading. The macro receives a single argument that corresponds to the heading text.

```
174 \def\markdownRendererHeadingTwo{%  
175   \markdownRendererHeadingTwoPrototype}%
```

The `\markdownRendererHeadingThree` macro represents a third level heading. The macro receives a single argument that corresponds to the heading text.

```
176 \def\markdownRendererHeadingThree{%  
177   \markdownRendererHeadingThreePrototype}%
```

The `\markdownRendererHeadingFour` macro represents a fourth level heading. The macro receives a single argument that corresponds to the heading text.

```
178 \def\markdownRendererHeadingFour{%  
179   \markdownRendererHeadingFourPrototype}%
```

The `\markdownRendererHeadingFive` macro represents a fifth level heading. The macro receives a single argument that corresponds to the heading text.

```
180 \def\markdownRendererHeadingFive{%  
181   \markdownRendererHeadingFivePrototype}%
```

The `\markdownRendererHeadingSix` macro represents a sixth level heading. The macro receives a single argument that corresponds to the heading text.

```
182 \def\markdownRendererHeadingSix{%  
183   \markdownRendererHeadingSixPrototype}%
```

2.2.3.17 Horizontal Rule Renderer The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

```
184 \def\markdownRendererHorizontalRule{%
185   \markdownRendererHorizontalRulePrototype}%
```

2.2.3.18 Footnote Renderer The `\markdownRendererFootnote` macro represents a footnote. This macro will only be produced, when the `footnotes` option is `true`. The macro receives a single argument that corresponds to the footnote text.

```
186 \def\markdownRendererFootnote{%
187   \markdownRendererFootnotePrototype}%
```

2.2.3.19 Parenthesized Citations Renderer The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the `citations` option is `true`. The macro receives the parameter `{⟨number of citations⟩}` followed by `⟨suppress author⟩{⟨prenote⟩}{⟨postnote⟩}{⟨name⟩}` repeated `⟨number of citations⟩` times. The `⟨suppress author⟩` parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```
188 \def\markdownRendererCite{%
189   \markdownRendererCitePrototype}%
```

2.2.3.20 Text Citations Renderer The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced, when the `citations` option is `true`. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```
190 \def\markdownRendererTextCite{%
191   \markdownRendererTextCitePrototype}%
```

2.2.4 Token Renderer Prototypes

The following \TeX macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the \LaTeX and \ConTeXt implementations (see sections 3.3 and 3.4).

```
192 \def\markdownRendererInterblockSeparatorPrototype{%
193 \def\markdownRendererLineBreakPrototype}%
194 \def\markdownRendererEllipsisPrototype}%
195 \def\markdownRendererNbspPrototype}%
196 \def\markdownRendererLeftBracePrototype}%
197 \def\markdownRendererRightBracePrototype}%
198 \def\markdownRendererDollarSignPrototype}%
199 \def\markdownRendererPercentSignPrototype}%
```

```

200 \def\markdownRenderAmpersandPrototype{}%
201 \def\markdownRenderUnderscorePrototype{}%
202 \def\markdownRenderHashPrototype{}%
203 \def\markdownRenderCircumflexPrototype{}%
204 \def\markdownRenderBackslashPrototype{}%
205 \def\markdownRenderTildePrototype{}%
206 \def\markdownRenderPipePrototype{}%
207 \def\markdownRenderCodeSpanPrototype#1{}%
208 \def\markdownRenderLinkPrototype#1#2#3#4{}%
209 \def\markdownRenderImagePrototype#1#2#3#4{}%
210 \def\markdownRenderContentBlockPrototype#1#2#3#4{}%
211 \def\markdownRenderContentBlockOnlineImagePrototype#1#2#3#4{}%
212 \def\markdownRenderContentBlockCodePrototype#1#2#3#4#5{}%
213 \def\markdownRenderUllBeginPrototype{}%
214 \def\markdownRenderUllBeginTightPrototype{}%
215 \def\markdownRenderUllItemPrototype{}%
216 \def\markdownRenderUllItemEndPrototype{}%
217 \def\markdownRenderUllEndPrototype{}%
218 \def\markdownRenderUllEndTightPrototype{}%
219 \def\markdownRenderOllBeginPrototype{}%
220 \def\markdownRenderOllBeginTightPrototype{}%
221 \def\markdownRenderOllItemPrototype{}%
222 \def\markdownRenderOllItemWithNumberPrototype#1{}%
223 \def\markdownRenderOllItemEndPrototype{}%
224 \def\markdownRenderOllEndPrototype{}%
225 \def\markdownRenderOllEndTightPrototype{}%
226 \def\markdownRenderDllBeginPrototype{}%
227 \def\markdownRenderDllBeginTightPrototype{}%
228 \def\markdownRenderDllItemPrototype#1{}%
229 \def\markdownRenderDllItemEndPrototype{}%
230 \def\markdownRenderDllDefinitionBeginPrototype{}%
231 \def\markdownRenderDllDefinitionEndPrototype{}%
232 \def\markdownRenderDllEndPrototype{}%
233 \def\markdownRenderDllEndTightPrototype{}%
234 \def\markdownRenderEmphasisPrototype#1{}%
235 \def\markdownRenderStrongEmphasisPrototype#1{}%
236 \def\markdownRenderBlockQuoteBeginPrototype{}%
237 \def\markdownRenderBlockQuoteEndPrototype{}%
238 \def\markdownRenderInputVerbatimPrototype#1{}%
239 \def\markdownRenderInputFencedCodePrototype#1#2{}%
240 \def\markdownRenderHeadingOnePrototype#1{}%
241 \def\markdownRenderHeadingTwoPrototype#1{}%
242 \def\markdownRenderHeadingThreePrototype#1{}%
243 \def\markdownRenderHeadingFourPrototype#1{}%
244 \def\markdownRenderHeadingFivePrototype#1{}%
245 \def\markdownRenderHeadingSixPrototype#1{}%
246 \def\markdownRenderHorizontalRulePrototype{}%

```



```

247 \def\markdownRendererFootnotePrototype#1{%
248 \def\markdownRendererCitePrototype#1{%
249 \def\markdownRendererTextCitePrototype#1{%

```

2.2.5 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros provide access to logging to the rest of the macros. Their first argument specifies the text of the info, warning, or error message.

```

250 \def\markdownInfo#1{%
251 \def\markdownWarning#1{%

```

The `\markdownError` macro receives a second argument that provides a help text suggesting a remedy to the error.

```

252 \def\markdownError#1#2{%

```

You may redefine these macros to redirect and process the info, warning, and error messages.

2.2.6 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a \TeX engine that does not support direct Lua access is starting to buffer a text. The plain \TeX implementation changes the category code of plain \TeX special characters to *other*, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```

253 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain \TeX special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```

254 \let\markdownReadAndConvert\relax
255 \begingroup

```

Locally swap the category code of the backslash symbol (`\`) with the pipe symbol (`|`). This is required in order that all the special symbols in the first argument of the `\markdownReadAndConvert` macro have the category code *other*.

```

256 \catcode'\|=0\catcode'\=12%
257 |gdef|markdownBegin{%
258   |markdownReadAndConvert{\markdownEnd}%
259   { |markdownEnd}}%
260 |endgroup

```

The macro is exposed in the interface, so that the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.6), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol `]`).

The `\markdownMode` macro specifies how the plain \TeX implementation interfaces with the Lua interface. The valid values and their meaning are as follows:

- 0 – Shell escape via the 18 output file stream
- 1 – Shell escape via the Lua `os.execute` method
- 2 – Direct Lua access

By defining the macro, the user can coerce the package to use a specific mode. If the user does not define the macro prior to loading the plain \TeX implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

```
261 \ifx\markdownMode\undefined
262   \ifx\directlua\undefined
263     \def\markdownMode{0}%
264   \else
265     \def\markdownMode{2}%
266   \fi
267 \fi
```

The following macros are no longer a part of the plain \TeX interface and are only defined for backwards compatibility:

```
268 \def\markdownLuaRegisterIBCallback#1{\relax}%
269 \def\markdownLuaUnregisterIBCallback#1{\relax}%
```

2.3 \LaTeX Interface

The \LaTeX interface provides \LaTeX environments for the typesetting of markdown input from within \LaTeX , facilities for setting Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain \TeX , and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain \TeX interface (see Section 2.2).

The \LaTeX interface is implemented by the `markdown.sty` file, which can be loaded from the \LaTeX document preamble as follows:

```
\usepackage[⟨options⟩]{markdown}
```

where `⟨options⟩` are the \LaTeX interface options (see Section 2.3.2). Note that `⟨options⟩` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.2) and `markdownRendererPrototypes` (see Section 2.3.2.3) keys. This limitation is due to the way $\LaTeX 2_{\epsilon}$ parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` \LaTeX environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` \LaTeX environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts \LaTeX interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
270 \newenvironment{markdown}\relax\relax
271 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` \LaTeX environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` \LaTeX environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain \TeX interface.

The following example \LaTeX code showcases the usage of the `markdown` and `markdown*` environments:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown} _Hello_ **world** ... \end{markdown} % ... \end{document}</pre>	<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ **world** ... \end{markdown*} % ... \end{document}</pre>
--	---

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain \TeX . Unlike the `\markdownInput` macro provided by the plain \TeX interface, this macro also accepts \LaTeX interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example \LaTeX code showcases the usage of the `\markdownInput` macro:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \markdownInput[smartEllipses]{hello.md}</pre>

```
% ...
\end{document}
```

2.3.2 Options

The \LaTeX options are represented by a comma-delimited list of $\langle\langle key \rangle = \langle value \rangle\rangle$ pairs. For boolean options, the $\langle = \langle value \rangle \rangle$ part is optional, and $\langle\langle key \rangle\rangle$ will be interpreted as $\langle\langle key \rangle = true \rangle$.

The \LaTeX options map directly to the options recognized by the plain \TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain \TeX interface (see Sections 2.2.3 and 2.2.4).

The \LaTeX options may be specified when loading the \LaTeX package (see Section 2.3), when using the `markdown*` \LaTeX environment, or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument.

```
272 \newcommand\markdownSetup[1]{%
273   \setkeys{markdownOptions}{#1}}%
```

2.3.2.1 Plain \TeX Interface Options The following options map directly to the option macros exposed by the plain \TeX interface (see Section 2.2.2).

```
274 \define@key{markdownOptions}{helperScriptFileName}{%
275   \def\markdownOptionHelperScriptFileName{#1}}%
276 \define@key{markdownOptions}{inputTempFileName}{%
277   \def\markdownOptionInputTempFileName{#1}}%
278 \define@key{markdownOptions}{outputTempFileName}{%
279   \def\markdownOptionOutputTempFileName{#1}}%
280 \define@key{markdownOptions}{errorTempFileName}{%
281   \def\markdownOptionErrorTempFileName{#1}}%
282 \define@key{markdownOptions}{cacheDir}{%
283   \def\markdownOptionCacheDir{#1}}%
284 \define@key{markdownOptions}{outputDir}{%
285   \def\markdownOptionOutputDir{#1}}%
286 \define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
287   \def\markdownOptionBlankBeforeBlockquote{#1}}%
288 \define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
289   \def\markdownOptionBlankBeforeCodeFence{#1}}%
290 \define@key{markdownOptions}{blankBeforeHeading}[true]{%
291   \def\markdownOptionBlankBeforeHeading{#1}}%
292 \define@key{markdownOptions}{breakableBlockquotes}[true]{%
293   \def\markdownOptionBreakableBlockquotes{#1}}%
294 \define@key{markdownOptions}{citations}[true]{%
295   \def\markdownOptionCitations{#1}}%
296 \define@key{markdownOptions}{citationNbsps}[true]{%
297   \def\markdownOptionCitationNbsps{#1}}%
298 \define@key{markdownOptions}{contentBlocks}[true]{%
```

```

299 \def\markdownOptionContentBlocks{#1}}%
300 \define@key{markdownOptions}{codeSpans}[true]{%
301 \def\markdownOptionCodeSpans{#1}}%
302 \define@key{markdownOptions}{contentBlocksLanguageMap}{%
303 \def\markdownOptionContentBlocksLanguageMap{#1}}%
304 \define@key{markdownOptions}{definitionLists}[true]{%
305 \def\markdownOptionDefinitionLists{#1}}%
306 \define@key{markdownOptions}{footnotes}[true]{%
307 \def\markdownOptionFootnotes{#1}}%
308 \define@key{markdownOptions}{fencedCode}[true]{%
309 \def\markdownOptionFencedCode{#1}}%
310 \define@key{markdownOptions}{hashEnumerators}[true]{%
311 \def\markdownOptionHashEnumerators{#1}}%
312 \define@key{markdownOptions}{html}[true]{%
313 \def\markdownOptionHtml{#1}}%
314 \define@key{markdownOptions}{hybrid}[true]{%
315 \def\markdownOptionHybrid{#1}}%
316 \define@key{markdownOptions}{inlineFootnotes}[true]{%
317 \def\markdownOptionInlineFootnotes{#1}}%
318 \define@key{markdownOptions}{preserveTabs}[true]{%
319 \def\markdownOptionPreserveTabs{#1}}%
320 \define@key{markdownOptions}{smartEllipses}[true]{%
321 \def\markdownOptionSmartEllipses{#1}}%
322 \define@key{markdownOptions}{startNumber}[true]{%
323 \def\markdownOptionStartNumber{#1}}%
324 \define@key{markdownOptions}{tightLists}[true]{%
325 \def\markdownOptionTightLists{#1}}%
326 \define@key{markdownOptions}{underscores}[true]{%
327 \def\markdownOptionUnderscores{#1}}%

```

The following example \LaTeX code showcases a possible configuration of plain \TeX interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

```

\markdownSetup{
  hybrid,
  smartEllipses,
  cacheDir = /tmp,
}

```

2.3.2.2 Plain \TeX Markdown Token Renderers The \LaTeX interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain \TeX interface (see Section 2.2.3).

```

328 \define@key{markdownRenderers}{interblockSeparator}{%

```

```

329 \renewcommand\markdownRendererInterblockSeparator{#1}}%
330 \define@key{markdownRenderers}{lineBreak}{%
331 \renewcommand\markdownRendererLineBreak{#1}}%
332 \define@key{markdownRenderers}{ellipsis}{%
333 \renewcommand\markdownRendererEllipsis{#1}}%
334 \define@key{markdownRenderers}{nbsp}{%
335 \renewcommand\markdownRendererNbsp{#1}}%
336 \define@key{markdownRenderers}{leftBrace}{%
337 \renewcommand\markdownRendererLeftBrace{#1}}%
338 \define@key{markdownRenderers}{rightBrace}{%
339 \renewcommand\markdownRendererRightBrace{#1}}%
340 \define@key{markdownRenderers}{dollarSign}{%
341 \renewcommand\markdownRendererDollarSign{#1}}%
342 \define@key{markdownRenderers}{percentSign}{%
343 \renewcommand\markdownRendererPercentSign{#1}}%
344 \define@key{markdownRenderers}{ampersand}{%
345 \renewcommand\markdownRendererAmpersand{#1}}%
346 \define@key{markdownRenderers}{underscore}{%
347 \renewcommand\markdownRendererUnderscore{#1}}%
348 \define@key{markdownRenderers}{hash}{%
349 \renewcommand\markdownRendererHash{#1}}%
350 \define@key{markdownRenderers}{circumflex}{%
351 \renewcommand\markdownRendererCircumflex{#1}}%
352 \define@key{markdownRenderers}{backslash}{%
353 \renewcommand\markdownRendererBackslash{#1}}%
354 \define@key{markdownRenderers}{tilde}{%
355 \renewcommand\markdownRendererTilde{#1}}%
356 \define@key{markdownRenderers}{pipe}{%
357 \renewcommand\markdownRendererPipe{#1}}%
358 \define@key{markdownRenderers}{codeSpan}{%
359 \renewcommand\markdownRendererCodeSpan[1]{#1}}%
360 \define@key{markdownRenderers}{link}{%
361 \renewcommand\markdownRendererLink[4]{#1}}%
362 \define@key{markdownRenderers}{contentBlock}{%
363 \renewcommand\markdownRendererContentBlock[4]{#1}}%
364 \define@key{markdownRenderers}{contentBlockOnlineImage}{%
365 \renewcommand\markdownRendererContentBlockOnlineImage[4]{#1}}%
366 \define@key{markdownRenderers}{contentBlockCode}{%
367 \renewcommand\markdownRendererContentBlockCode[5]{#1}}%
368 \define@key{markdownRenderers}{image}{%
369 \renewcommand\markdownRendererImage[4]{#1}}%
370 \define@key{markdownRenderers}{ulBegin}{%
371 \renewcommand\markdownRendererUlBegin{#1}}%
372 \define@key{markdownRenderers}{ulBeginTight}{%
373 \renewcommand\markdownRendererUlBeginTight{#1}}%
374 \define@key{markdownRenderers}{ulItem}{%
375 \renewcommand\markdownRendererUlItem{#1}}%

```

```

376 \define@key{markdownRenderers}{ulItemEnd}{%
377   \renewcommand\markdownRendererUlItemEnd{#1}}%
378 \define@key{markdownRenderers}{ulEnd}{%
379   \renewcommand\markdownRendererUlEnd{#1}}%
380 \define@key{markdownRenderers}{ulEndTight}{%
381   \renewcommand\markdownRendererUlEndTight{#1}}%
382 \define@key{markdownRenderers}{olBegin}{%
383   \renewcommand\markdownRendererOlBegin{#1}}%
384 \define@key{markdownRenderers}{olBeginTight}{%
385   \renewcommand\markdownRendererOlBeginTight{#1}}%
386 \define@key{markdownRenderers}{olItem}{%
387   \renewcommand\markdownRendererOlItem{#1}}%
388 \define@key{markdownRenderers}{olItemWithNumber}{%
389   \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
390 \define@key{markdownRenderers}{olItemEnd}{%
391   \renewcommand\markdownRendererOlItemEnd{#1}}%
392 \define@key{markdownRenderers}{olEnd}{%
393   \renewcommand\markdownRendererOlEnd{#1}}%
394 \define@key{markdownRenderers}{olEndTight}{%
395   \renewcommand\markdownRendererOlEndTight{#1}}%
396 \define@key{markdownRenderers}{dlBegin}{%
397   \renewcommand\markdownRendererDlBegin{#1}}%
398 \define@key{markdownRenderers}{dlBeginTight}{%
399   \renewcommand\markdownRendererDlBeginTight{#1}}%
400 \define@key{markdownRenderers}{dlItem}{%
401   \renewcommand\markdownRendererDlItem[1]{#1}}%
402 \define@key{markdownRenderers}{dlItemEnd}{%
403   \renewcommand\markdownRendererDlItemEnd{#1}}%
404 \define@key{markdownRenderers}{dlDefinitionBegin}{%
405   \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
406 \define@key{markdownRenderers}{dlDefinitionEnd}{%
407   \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
408 \define@key{markdownRenderers}{dlEnd}{%
409   \renewcommand\markdownRendererDlEnd{#1}}%
410 \define@key{markdownRenderers}{dlEndTight}{%
411   \renewcommand\markdownRendererDlEndTight{#1}}%
412 \define@key{markdownRenderers}{emphasis}{%
413   \renewcommand\markdownRendererEmphasis[1]{#1}}%
414 \define@key{markdownRenderers}{strongEmphasis}{%
415   \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
416 \define@key{markdownRenderers}{blockquoteBegin}{%
417   \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
418 \define@key{markdownRenderers}{blockquoteEnd}{%
419   \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
420 \define@key{markdownRenderers}{inputVerbatim}{%
421   \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
422 \define@key{markdownRenderers}{inputFencedCode}{%

```

```

423 \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
424 \define@key{markdownRenderers}{headingOne}{%
425 \renewcommand\markdownRendererHeadingOne[1]{#1}}%
426 \define@key{markdownRenderers}{headingTwo}{%
427 \renewcommand\markdownRendererHeadingTwo[1]{#1}}%
428 \define@key{markdownRenderers}{headingThree}{%
429 \renewcommand\markdownRendererHeadingThree[1]{#1}}%
430 \define@key{markdownRenderers}{headingFour}{%
431 \renewcommand\markdownRendererHeadingFour[1]{#1}}%
432 \define@key{markdownRenderers}{headingFive}{%
433 \renewcommand\markdownRendererHeadingFive[1]{#1}}%
434 \define@key{markdownRenderers}{headingSix}{%
435 \renewcommand\markdownRendererHeadingSix[1]{#1}}%
436 \define@key{markdownRenderers}{horizontalRule}{%
437 \renewcommand\markdownRendererHorizontalRule{#1}}%
438 \define@key{markdownRenderers}{footnote}{%
439 \renewcommand\markdownRendererFootnote[1]{#1}}%
440 \define@key{markdownRenderers}{cite}{%
441 \renewcommand\markdownRendererCite[1]{#1}}%
442 \define@key{markdownRenderers}{textCite}{%
443 \renewcommand\markdownRendererTextCite[1]{#1}}%

```

The following example \LaTeX code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```

\markdownSetup{
  renderers = {
    link = {#4}, % Render links as the link title.
    emphasis = {\emph{#1}}, % Render emphasized text via '\emph'.
  }
}

```

2.3.2.3 Plain \LaTeX Markdown Token Renderer Prototypes The \LaTeX interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain \LaTeX interface (see Section 2.2.4).

```

444 \define@key{markdownRendererPrototypes}{interblockSeparator}{%
445 \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
446 \define@key{markdownRendererPrototypes}{lineBreak}{%
447 \renewcommand\markdownRendererLineBreakPrototype{#1}}%
448 \define@key{markdownRendererPrototypes}{ellipsis}{%
449 \renewcommand\markdownRendererEllipsisPrototype{#1}}%
450 \define@key{markdownRendererPrototypes}{nbsp}{%
451 \renewcommand\markdownRendererNbspPrototype{#1}}%

```



```

452 \define@key{markdownRendererPrototypes}{leftBrace}{%
453   \renewcommand\markdownRendererLeftBracePrototype{#1}}%
454 \define@key{markdownRendererPrototypes}{rightBrace}{%
455   \renewcommand\markdownRendererRightBracePrototype{#1}}%
456 \define@key{markdownRendererPrototypes}{dollarSign}{%
457   \renewcommand\markdownRendererDollarSignPrototype{#1}}%
458 \define@key{markdownRendererPrototypes}{percentSign}{%
459   \renewcommand\markdownRendererPercentSignPrototype{#1}}%
460 \define@key{markdownRendererPrototypes}{ampersand}{%
461   \renewcommand\markdownRendererAmpersandPrototype{#1}}%
462 \define@key{markdownRendererPrototypes}{underscore}{%
463   \renewcommand\markdownRendererUnderscorePrototype{#1}}%
464 \define@key{markdownRendererPrototypes}{hash}{%
465   \renewcommand\markdownRendererHashPrototype{#1}}%
466 \define@key{markdownRendererPrototypes}{circumflex}{%
467   \renewcommand\markdownRendererCircumflexPrototype{#1}}%
468 \define@key{markdownRendererPrototypes}{backslash}{%
469   \renewcommand\markdownRendererBackslashPrototype{#1}}%
470 \define@key{markdownRendererPrototypes}{tilde}{%
471   \renewcommand\markdownRendererTildePrototype{#1}}%
472 \define@key{markdownRendererPrototypes}{pipe}{%
473   \renewcommand\markdownRendererPipePrototype{#1}}%
474 \define@key{markdownRendererPrototypes}{codeSpan}{%
475   \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
476 \define@key{markdownRendererPrototypes}{link}{%
477   \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
478 \define@key{markdownRendererPrototypes}{contentBlock}{%
479   \renewcommand\markdownRendererContentBlockPrototype[4]{#1}}%
480 \define@key{markdownRendererPrototypes}{contentBlockOnlineImage}{%
481   \renewcommand\markdownRendererContentBlockOnlineImagePrototype[4]{#1}}%
482 \define@key{markdownRendererPrototypes}{contentBlockCode}{%
483   \renewcommand\markdownRendererContentBlockCodePrototype[5]{#1}}%
484 \define@key{markdownRendererPrototypes}{image}{%
485   \renewcommand\markdownRendererImagePrototype[4]{#1}}%
486 \define@key{markdownRendererPrototypes}{ulBegin}{%
487   \renewcommand\markdownRendererUlBeginPrototype{#1}}%
488 \define@key{markdownRendererPrototypes}{ulBeginTight}{%
489   \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
490 \define@key{markdownRendererPrototypes}{ulItem}{%
491   \renewcommand\markdownRendererUlItemPrototype{#1}}%
492 \define@key{markdownRendererPrototypes}{ulItemEnd}{%
493   \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
494 \define@key{markdownRendererPrototypes}{ulEnd}{%
495   \renewcommand\markdownRendererUlEndPrototype{#1}}%
496 \define@key{markdownRendererPrototypes}{ulEndTight}{%
497   \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
498 \define@key{markdownRendererPrototypes}{olBegin}{%

```

```

499 \renewcommand\markdownRendererOlBeginPrototype{#1}}%
500 \define@key{markdownRendererPrototypes}{olBeginTight}{%
501 \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
502 \define@key{markdownRendererPrototypes}{olItem}{%
503 \renewcommand\markdownRendererOlItemPrototype{#1}}%
504 \define@key{markdownRendererPrototypes}{olItemWithNumber}{%
505 \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
506 \define@key{markdownRendererPrototypes}{olItemEnd}{%
507 \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
508 \define@key{markdownRendererPrototypes}{olEnd}{%
509 \renewcommand\markdownRendererOlEndPrototype{#1}}%
510 \define@key{markdownRendererPrototypes}{olEndTight}{%
511 \renewcommand\markdownRendererOlEndTightPrototype{#1}}%
512 \define@key{markdownRendererPrototypes}{dlBegin}{%
513 \renewcommand\markdownRendererDlBeginPrototype{#1}}%
514 \define@key{markdownRendererPrototypes}{dlBeginTight}{%
515 \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
516 \define@key{markdownRendererPrototypes}{dlItem}{%
517 \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
518 \define@key{markdownRendererPrototypes}{dlItemEnd}{%
519 \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
520 \define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
521 \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%
522 \define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
523 \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
524 \define@key{markdownRendererPrototypes}{dlEnd}{%
525 \renewcommand\markdownRendererDlEndPrototype{#1}}%
526 \define@key{markdownRendererPrototypes}{dlEndTight}{%
527 \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
528 \define@key{markdownRendererPrototypes}{emphasis}{%
529 \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
530 \define@key{markdownRendererPrototypes}{strongEmphasis}{%
531 \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
532 \define@key{markdownRendererPrototypes}{blockQuoteBegin}{%
533 \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
534 \define@key{markdownRendererPrototypes}{blockQuoteEnd}{%
535 \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
536 \define@key{markdownRendererPrototypes}{inputVerbatim}{%
537 \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
538 \define@key{markdownRendererPrototypes}{inputFencedCode}{%
539 \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
540 \define@key{markdownRendererPrototypes}{headingOne}{%
541 \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
542 \define@key{markdownRendererPrototypes}{headingTwo}{%
543 \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
544 \define@key{markdownRendererPrototypes}{headingThree}{%
545 \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%

```

```

546 \define@key{markdownRendererPrototypes}{headingFour}{%
547   \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%
548 \define@key{markdownRendererPrototypes}{headingFive}{%
549   \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
550 \define@key{markdownRendererPrototypes}{headingSix}{%
551   \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
552 \define@key{markdownRendererPrototypes}{horizontalRule}{%
553   \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
554 \define@key{markdownRendererPrototypes}{footnote}{%
555   \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
556 \define@key{markdownRendererPrototypes}{cite}{%
557   \renewcommand\markdownRendererCitePrototype[1]{#1}}%
558 \define@key{markdownRendererPrototypes}{textCite}{%
559   \renewcommand\markdownRendererTextCitePrototype[1]{#1}}%

```

The following example \TeX code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```

\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}},    % Render inline code via '\texttt'.
  }
}

```

2.4 Con \TeX Interface

The Con \TeX interface provides a start-stop macro pair for the typesetting of markdown input from within Con \TeX . The rest of the interface is inherited from the plain \TeX interface (see Section 2.2).

```

560 \writestatus{loading}{ConTeXt User Module / markdown}%
561 \unprotect

```

The Con \TeX interface is implemented by the `t-markdown.tex` Con \TeX module file that can be loaded as follows:

```

\usemodule[t] [markdown]

```

It is expected that the special plain \TeX characters have the expected category codes, when `\inputting` the file.

2.4.1 Typesetting Markdown

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment.

```
562 \let\startmarkdown\relax
563 \let\stopmarkdown\relax
```

You may prepend your own code to the `\startmarkdown` macro and redefine the `\stopmarkdown` macro to produce special effects before and after the markdown block.

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain \TeX interface.

The following example Con \TeX t code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```
\usemodule[t][markdown]
\starttext
\startmarkdown
_Hello_ world ...
\stopmarkdown
\stoptext
```

3 Technical Documentation

This part of the manual describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects that provide the conversion from markdown to plain \TeX .

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain \TeX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
564 local upper, gsub, format, length =
565   string.upper, string.gsub, string.format, string.len
566 local concat = table.concat
567 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
568   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
569   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain T_EX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
570 local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

```
571 function util.err(msg, exit_code)
572   io.stderr:write("markdown.lua: " .. msg .. "\n")
573   os.exit(exit_code or 1)
574 end
```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
575 function util.cache(dir, string, salt, transform, suffix)
576   local digest = md5.sumhexa(string .. (salt or ""))
577   local name = util.pathname(dir, digest .. suffix)
578   local file = io.open(name, "r")
579   if file == nil then -- If no cache entry exists, then create a new one.
580     local file = assert(io.open(name, "w"))
581     local result = string
582     if transform ~= nil then
583       result = transform(result)
584     end
585     assert(file:write(result))
586     assert(file:close())
587   end
588   return name
589 end
```

The `util.table_copy` method creates a shallow copy of a table `t` and its metatable.

```
590 function util.table_copy(t)
591   local u = { }
592   for k, v in pairs(t) do u[k] = v end
593   return setmetatable(u, getmetatable(t))
594 end
```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from [4, Chapter 21].

```
595 function util.expand_tabs_in_line(s, tabstop)
596   local tab = tabstop or 4
597   local corr = 0
598   return (s:gsub("()\t", function(p)
```

```

599         local sp = tab - (p - 1 + corr) % tab
600         corr = corr - 1 + sp
601         return string.rep(" ", sp)
602     end))
603 end

```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```

604 function util.walk(t, f)
605     local typ = type(t)
606     if typ == "string" then
607         f(t)
608     elseif typ == "table" then
609         local i = 1
610         local n
611         n = t[i]
612         while n do
613             util.walk(n, f)
614             i = i + 1
615             n = t[i]
616         end
617     elseif typ == "function" then
618         local ok, val = pcall(t)
619         if ok then
620             util.walk(val, f)
621         end
622     else
623         f(tostring(t))
624     end
625 end

```

The `util.flatten` method flattens an array `ary` that does not contain cycles and returns the result.

```

626 function util.flatten(ary)
627     local new = {}
628     for _,v in ipairs(ary) do
629         if type(v) == "table" then
630             for _,w in ipairs(util.flatten(v)) do
631                 new[#new + 1] = w
632             end
633         else
634             new[#new + 1] = v
635         end
636     end
637     return new
638 end

```

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

```
639 function util.rope_to_string(rope)
640   local buffer = {}
641   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
642   return table.concat(buffer)
643 end
```

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```
644 function util.rope_last(rope)
645   if #rope == 0 then
646     return nil
647   else
648     local l = rope[#rope]
649     if type(l) == "table" then
650       return util.rope_last(l)
651     else
652       return l
653     end
654   end
655 end
```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all $1 \leq i \leq \#ary$.

```
656 function util.intersperse(ary, x)
657   local new = {}
658   local l = #ary
659   for i,v in ipairs(ary) do
660     local n = #new
661     new[n + 1] = v
662     if i ~= l then
663       new[n + 2] = x
664     end
665   end
666   return new
667 end
```

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all $1 \leq i \leq \#ary$.

```
668 function util.map(ary, f)
669   local new = {}
670   for i,v in ipairs(ary) do
671     new[i] = f(v)
672   end
673   return new
674 end
```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurrences of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

```
675 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
676 local char_escapes_list = ""
677 for i,_ in pairs(char_escapes) do
678     char_escapes_list = char_escapes_list .. i
679 end
```

Create an LPeg capture `escapable` that produces the escaped string corresponding to the matched escapable character.

```
680 local escapable = S(char_escapes_list) / char_escapes
```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{(k,v) \in \text{string_escapes}} P(k) / v + \text{escapable}$$

capture that replaces any occurrence of the string `k` with the string `v` for each $(k, v) \in \text{string_escapes}$. Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corollary, the strings always take precedence over the characters.

```
681 if string_escapes then
682     for k,v in pairs(string_escapes) do
683         escapable = P(k) / v + escapable
684     end
685 end
```

Create an LPeg capture `escape_string` that captures anything `escapable` does and matches any other unmatched characters.

```
686 local escape_string = Cs((escapable + any)^0)
```

Return a function that matches the input string `s` against the `escape_string` capture.

```
687 return function(s)
688     return lpeg.match(escape_string, s)
689 end
690 end
```

The `util.pathname` method produces a pathname out of a directory name `dir` and a filename `file` and returns it.

```
691 function util.pathname(dir, file)
692     if #dir == 0 then
693         return file
694     else
```



```

695     return dir .. "/" .. file
696 end
697 end

```

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```

698 local entities = {}
699
700 local character_entities = {
701     ["quot"] = 0x0022,
702     ["amp"] = 0x0026,
703     ["apos"] = 0x0027,
704     ["lt"] = 0x003C,
705     ["gt"] = 0x003E,
706     ["nbsp"] = 160,
707     ["iexcl"] = 0x00A1,
708     ["cent"] = 0x00A2,
709     ["pound"] = 0x00A3,
710     ["curren"] = 0x00A4,
711     ["yen"] = 0x00A5,
712     ["brvbar"] = 0x00A6,
713     ["sect"] = 0x00A7,
714     ["uml"] = 0x00A8,
715     ["copy"] = 0x00A9,
716     ["ordf"] = 0x00AA,
717     ["laquo"] = 0x00AB,
718     ["not"] = 0x00AC,
719     ["shy"] = 173,
720     ["reg"] = 0x00AE,
721     ["macr"] = 0x00AF,
722     ["deg"] = 0x00B0,
723     ["plusmn"] = 0x00B1,
724     ["sup2"] = 0x00B2,
725     ["sup3"] = 0x00B3,
726     ["acute"] = 0x00B4,
727     ["micro"] = 0x00B5,
728     ["para"] = 0x00B6,
729     ["middot"] = 0x00B7,
730     ["cedil"] = 0x00B8,
731     ["sup1"] = 0x00B9,
732     ["ordm"] = 0x00BA,
733     ["raquo"] = 0x00BB,
734     ["frac14"] = 0x00BC,

```

```

735 ["frac12"] = 0x00BD,
736 ["frac34"] = 0x00BE,
737 ["iquest"] = 0x00BF,
738 ["Agrave"] = 0x00C0,
739 ["Aacute"] = 0x00C1,
740 ["Acirc"] = 0x00C2,
741 ["Atilde"] = 0x00C3,
742 ["Auml"] = 0x00C4,
743 ["Aring"] = 0x00C5,
744 ["AElig"] = 0x00C6,
745 ["Ccedil"] = 0x00C7,
746 ["Egrave"] = 0x00C8,
747 ["Eacute"] = 0x00C9,
748 ["Ecirc"] = 0x00CA,
749 ["Euml"] = 0x00CB,
750 ["Igrave"] = 0x00CC,
751 ["Iacute"] = 0x00CD,
752 ["Icirc"] = 0x00CE,
753 ["Iuml"] = 0x00CF,
754 ["ETH"] = 0x00D0,
755 ["Ntilde"] = 0x00D1,
756 ["Ograve"] = 0x00D2,
757 ["Oacute"] = 0x00D3,
758 ["Ocirc"] = 0x00D4,
759 ["Otilde"] = 0x00D5,
760 ["Ouml"] = 0x00D6,
761 ["times"] = 0x00D7,
762 ["Oslash"] = 0x00D8,
763 ["Ugrave"] = 0x00D9,
764 ["Uacute"] = 0x00DA,
765 ["Ucirc"] = 0x00DB,
766 ["Uuml"] = 0x00DC,
767 ["Yacute"] = 0x00DD,
768 ["THORN"] = 0x00DE,
769 ["szlig"] = 0x00DF,
770 ["agrave"] = 0x00E0,
771 ["aacute"] = 0x00E1,
772 ["acirc"] = 0x00E2,
773 ["atilde"] = 0x00E3,
774 ["auml"] = 0x00E4,
775 ["aring"] = 0x00E5,
776 ["aelig"] = 0x00E6,
777 ["ccedil"] = 0x00E7,
778 ["egrave"] = 0x00E8,
779 ["eacute"] = 0x00E9,
780 ["ecirc"] = 0x00EA,
781 ["euml"] = 0x00EB,

```

```

782 ["igrave"] = 0x00EC,
783 ["iacute"] = 0x00ED,
784 ["icirc"] = 0x00EE,
785 ["iuml"] = 0x00EF,
786 ["eth"] = 0x00F0,
787 ["ntilde"] = 0x00F1,
788 ["ograve"] = 0x00F2,
789 ["oacute"] = 0x00F3,
790 ["ocirc"] = 0x00F4,
791 ["otilde"] = 0x00F5,
792 ["ouml"] = 0x00F6,
793 ["divide"] = 0x00F7,
794 ["oslash"] = 0x00F8,
795 ["ugrave"] = 0x00F9,
796 ["uacute"] = 0x00FA,
797 ["ucirc"] = 0x00FB,
798 ["uuml"] = 0x00FC,
799 ["yacute"] = 0x00FD,
800 ["thorn"] = 0x00FE,
801 ["yuml"] = 0x00FF,
802 ["OElig"] = 0x0152,
803 ["oelig"] = 0x0153,
804 ["Scaron"] = 0x0160,
805 ["scaron"] = 0x0161,
806 ["Yuml"] = 0x0178,
807 ["fnof"] = 0x0192,
808 ["circ"] = 0x02C6,
809 ["tilde"] = 0x02DC,
810 ["Alpha"] = 0x0391,
811 ["Beta"] = 0x0392,
812 ["Gamma"] = 0x0393,
813 ["Delta"] = 0x0394,
814 ["Epsilon"] = 0x0395,
815 ["Zeta"] = 0x0396,
816 ["Eta"] = 0x0397,
817 ["Theta"] = 0x0398,
818 ["Iota"] = 0x0399,
819 ["Kappa"] = 0x039A,
820 ["Lambda"] = 0x039B,
821 ["Mu"] = 0x039C,
822 ["Nu"] = 0x039D,
823 ["Xi"] = 0x039E,
824 ["Omicron"] = 0x039F,
825 ["Pi"] = 0x03A0,
826 ["Rho"] = 0x03A1,
827 ["Sigma"] = 0x03A3,
828 ["Tau"] = 0x03A4,

```

```

829 ["Upsilon"] = 0x03A5,
830 ["Phi"] = 0x03A6,
831 ["Chi"] = 0x03A7,
832 ["Psi"] = 0x03A8,
833 ["Omega"] = 0x03A9,
834 ["alpha"] = 0x03B1,
835 ["beta"] = 0x03B2,
836 ["gamma"] = 0x03B3,
837 ["delta"] = 0x03B4,
838 ["epsilon"] = 0x03B5,
839 ["zeta"] = 0x03B6,
840 ["eta"] = 0x03B7,
841 ["theta"] = 0x03B8,
842 ["iota"] = 0x03B9,
843 ["kappa"] = 0x03BA,
844 ["lambda"] = 0x03BB,
845 ["mu"] = 0x03BC,
846 ["nu"] = 0x03BD,
847 ["xi"] = 0x03BE,
848 ["omicron"] = 0x03BF,
849 ["pi"] = 0x03C0,
850 ["rho"] = 0x03C1,
851 ["sigmaf"] = 0x03C2,
852 ["sigma"] = 0x03C3,
853 ["tau"] = 0x03C4,
854 ["upsilon"] = 0x03C5,
855 ["phi"] = 0x03C6,
856 ["chi"] = 0x03C7,
857 ["psi"] = 0x03C8,
858 ["omega"] = 0x03C9,
859 ["thetasym"] = 0x03D1,
860 ["upsih"] = 0x03D2,
861 ["piv"] = 0x03D6,
862 ["ensp"] = 0x2002,
863 ["emsp"] = 0x2003,
864 ["thinsp"] = 0x2009,
865 ["ndash"] = 0x2013,
866 ["mdash"] = 0x2014,
867 ["lsquo"] = 0x2018,
868 ["rsquo"] = 0x2019,
869 ["sbquo"] = 0x201A,
870 ["ldquo"] = 0x201C,
871 ["rdquo"] = 0x201D,
872 ["bdquo"] = 0x201E,
873 ["dagger"] = 0x2020,
874 ["Dagger"] = 0x2021,
875 ["bull"] = 0x2022,

```

```

876 ["hellip"] = 0x2026,
877 ["permil"] = 0x2030,
878 ["prime"] = 0x2032,
879 ["Prime"] = 0x2033,
880 ["lsaquo"] = 0x2039,
881 ["rsaquo"] = 0x203A,
882 ["oline"] = 0x203E,
883 ["frasl"] = 0x2044,
884 ["euro"] = 0x20AC,
885 ["image"] = 0x2111,
886 ["weierp"] = 0x2118,
887 ["real"] = 0x211C,
888 ["trade"] = 0x2122,
889 ["alefsym"] = 0x2135,
890 ["larr"] = 0x2190,
891 ["uarr"] = 0x2191,
892 ["rarr"] = 0x2192,
893 ["darr"] = 0x2193,
894 ["harr"] = 0x2194,
895 ["crarr"] = 0x21B5,
896 ["lArr"] = 0x21D0,
897 ["uArr"] = 0x21D1,
898 ["rArr"] = 0x21D2,
899 ["dArr"] = 0x21D3,
900 ["hArr"] = 0x21D4,
901 ["forall"] = 0x2200,
902 ["part"] = 0x2202,
903 ["exist"] = 0x2203,
904 ["empty"] = 0x2205,
905 ["nabla"] = 0x2207,
906 ["isin"] = 0x2208,
907 ["notin"] = 0x2209,
908 ["ni"] = 0x220B,
909 ["prod"] = 0x220F,
910 ["sum"] = 0x2211,
911 ["minus"] = 0x2212,
912 ["lowast"] = 0x2217,
913 ["radic"] = 0x221A,
914 ["prop"] = 0x221D,
915 ["infin"] = 0x221E,
916 ["ang"] = 0x2220,
917 ["and"] = 0x2227,
918 ["or"] = 0x2228,
919 ["cap"] = 0x2229,
920 ["cup"] = 0x222A,
921 ["int"] = 0x222B,
922 ["there4"] = 0x2234,

```

```

923 ["sim"] = 0x223C,
924 ["cong"] = 0x2245,
925 ["asymp"] = 0x2248,
926 ["ne"] = 0x2260,
927 ["equiv"] = 0x2261,
928 ["le"] = 0x2264,
929 ["ge"] = 0x2265,
930 ["sub"] = 0x2282,
931 ["sup"] = 0x2283,
932 ["nsup"] = 0x2284,
933 ["sube"] = 0x2286,
934 ["supe"] = 0x2287,
935 ["oplus"] = 0x2295,
936 ["otimes"] = 0x2297,
937 ["perp"] = 0x22A5,
938 ["sdot"] = 0x22C5,
939 ["lceil"] = 0x2308,
940 ["rceil"] = 0x2309,
941 ["lfloor"] = 0x230A,
942 ["rfloor"] = 0x230B,
943 ["lang"] = 0x27E8,
944 ["rang"] = 0x27E9,
945 ["loz"] = 0x25CA,
946 ["spades"] = 0x2660,
947 ["clubs"] = 0x2663,
948 ["hearts"] = 0x2665,
949 ["diams"] = 0x2666,
950 }

```

Given a string `s` of decimal digits, the `entities.dec_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

951 function entities.dec_entity(s)
952   return unicode.utf8.char(tonumber(s))
953 end

```

Given a string `s` of hexadecimal digits, the `entities.hex_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

954 function entities.hex_entity(s)
955   return unicode.utf8.char(tonumber("0x"..s))
956 end

```

Given a character entity name `s` (like `ouml`), the `entities.char_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

957 function entities.char_entity(s)
958   local n = character_entities[s]
959   return unicode.utf8.char(n)
960 end

```

3.1.3 Plain T_EX Writer

This section documents the `writer` object, which implements the routines for producing the T_EX output. The object is an amalgamate of the generic, T_EX, and L^AT_EX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `writer` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `writer.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

```
961 M.writer = {}
```

The `writer.new` method creates and returns a new T_EX writer object associated with the Lua interface options (see Section 2.1.2) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `writer->member`.

```
962 function M.writer.new(options)
963   local self = {}
964   options = options or {}
```

Make the `options` table inherit from the `defaultOptions` table.

```
965   setmetatable(options, { __index = function (_, key)
966     return defaultOptions[key] end })
```

Define `writer->suffix` as the suffix of the produced cache files.

```
967   self.suffix = ".tex"
```

Define `writer->space` as the output format of a space character.

```
968   self.space = " "
```

Define `writer->nbsp` as the output format of a non-breaking space character.

```
969   self.nbsp = "\\markdownRendererNbsp{}"
```

Define `writer->plain` as a function that will transform an input plain text block `s` to the output format.

```
970   function self.plain(s)
971     return s
972   end
```

Define `writer->paragraph` as a function that will transform an input paragraph `s` to the output format.

```
973   function self.paragraph(s)
974     return s
975   end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
976 function self.pack(name)
977   return [[\input"']] .. name .. [[\"relax]]
978 end
```

Define `writer->interblocksep` as the output format of a block element separator.

```
979 self.interblocksep = "\\markdownRendererInterblockSeparator\n{"
```

Define `writer->eof` as the end of file marker in the output format.

```
980 self.eof = [[\relax]]
```

Define `writer->linebreak` as the output format of a forced line break.

```
981 self.linebreak = "\\markdownRendererLineBreak\n{"
```

Define `writer->ellipsis` as the output format of an ellipsis.

```
982 self.ellipsis = "\\markdownRendererEllipsis{"
```

Define `writer->hrule` as the output format of a horizontal rule.

```
983 self.hrule = "\\markdownRendererHorizontalRule{"
```

Define a table `escaped_chars` containing the mapping from special plain \TeX characters (including the active pipe character (`|`) of \ConTeXt) to their escaped variants. Define tables `escaped_minimal_chars` and `escaped_minimal_strings` containing the mapping from special plain characters and character strings that need to be escaped even in content that will not be typeset.

```
984 local escaped_chars = {
985   [{""] = "\\markdownRendererLeftBrace{"},
986   ["}"] = "\\markdownRendererRightBrace{"},
987   ["$"] = "\\markdownRendererDollarSign{"},
988   ["%"] = "\\markdownRendererPercentSign{"},
989   ["&"] = "\\markdownRendererAmpersand{"},
990   ["_"] = "\\markdownRendererUnderscore{"},
991   ["#"] = "\\markdownRendererHash{"},
992   ["^"] = "\\markdownRendererCircumflex{"},
993   ["\\"] = "\\markdownRendererBackslash{"},
994   ["~"] = "\\markdownRendererTilde{"},
995   ["|"] = "\\markdownRendererPipe{"},
996 }
997 local escaped_uri_chars = {
998   [{""] = "\\markdownRendererLeftBrace{"},
999   ["}"] = "\\markdownRendererRightBrace{"},
1000   ["%"] = "\\markdownRendererPercentSign{"},
1001   ["\\"] = "\\markdownRendererBackslash{"},
1002 }
1003 local escaped_citation_chars = {
1004   [{""] = "\\markdownRendererLeftBrace{"},
1005   ["}"] = "\\markdownRendererRightBrace{"},
1006   ["%"] = "\\markdownRendererPercentSign{"},
```



```

1007     ["#"] = "\\markdownRendererHash{}",
1008     ["\\"] = "\\markdownRendererBackslash{}",
1009   }
1010   local escaped_minimal_strings = {
1011     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
1012   }

```

Use the `escaped_chars` table to create an escaper function `escape` and the `escaped_minimal_chars` and `escaped_minimal_strings` tables to create an escaper function `escape_minimal`.

```

1013   local escape = util.escaper(escaped_chars)
1014   local escape_citation = util.escaper(escaped_citation_chars,
1015     escaped_minimal_strings)
1016   local escape_uri = util.escaper(escaped_uri_chars, escaped_minimal_strings)

```

Define `writer->string` as a function that will transform an input plain text span `s` to the output format and `writer->uri` as a function that will transform an input URI `u` to the output format. If the `hybrid` option is `true`, use identity functions. Otherwise, use the `escape` and `escape_minimal` functions.

```

1017   if options.hybrid then
1018     self.string = function(s) return s end
1019     self.citation = function(c) return c end
1020     self.uri = function(u) return u end
1021   else
1022     self.string = escape
1023     self.citation = escape_citation
1024     self.uri = escape_uri
1025   end

```

Define `writer->code` as a function that will transform an input inlined code span `s` to the output format.

```

1026   function self.code(s)
1027     return {"\\markdownRendererCodeSpan", escape(s), ""}
1028   end

```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, and `tit` to the title of the link.

```

1029   function self.link(lab,src,tit)
1030     return {"\\markdownRendererLink{" ,lab,""},
1031       {"",self.string(src),""},
1032       {"",self.uri(src),""},
1033       {"",self.string(tit or ""),""}}
1034   end

```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, and `tit` to the title of the image.

```

1035 function self.image(lab,src,tit)
1036     return {"\\markdownRendererImage{",lab,"}",
1037             {"",self.string(src),""},
1038             {"",self.uri(src),""},
1039             {"",self.string(tit or ""),"{}"}
1040 end

```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `options.contentBlocksLanguageMap` files located by `kpathsea` are loaded into a chain of tables. `languages_json` corresponds to the first table and is chained with the rest via Lua metatables.

```

1041 local languages_json = (function()
1042     local kpse = require('kpse')
1043     kpse.set_program_name('luatex')
1044     local base, prev, curr
1045     for _, file in ipairs{kpse.lookup(options.contentBlocksLanguageMap,
1046                                   { all=true })} do
1047         json = assert(io.open(file, "r")):read("*all")
1048                 :gsub('("[^\\n]-"):',',', '[%1]='')
1049         curr = (function()
1050             local _ENV={ json=json, load=load } -- run in sandbox
1051             return load("return"..json)()
1052         end)()
1053         if type(curr) == "table" then
1054             if base == nil then
1055                 base = curr
1056             else
1057                 setmetatable(prev, { __index = curr })
1058             end
1059             prev = curr
1060         end
1061     end
1062     return base or {}
1063 end)()

```

Define `writer->contentblock` as a function that will transform an input `iA Writer` content block to the output format, where `src` corresponds to the URI prefix, `suf` to the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```

1064 function self.contentblock(src,suf,type,tit)
1065     src = src.." "..suf
1066     suf = suf:lower()
1067     if type == "onlineimage" then
1068         return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
1069                 {"",self.string(src),""},
1070                 {"",self.uri(src),""},
1071                 {"",self.string(tit or ""),"{}"}

```

```

1072     elsif languages_json[suf] then
1073         return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
1074               "{" ,self.string(languages_json[suf]),"} ",
1075               "{" ,self.string(src),"} ",
1076               "{" ,self.uri(src),"} ",
1077               "{" ,self.string(tit or ""),"} "}
1078     else
1079         return {"\\markdownRendererContentBlock{" ,suf,"} ",
1080               "{" ,self.string(src),"} ",
1081               "{" ,self.uri(src),"} ",
1082               "{" ,self.string(tit or ""),"} "}
1083     end
1084 end

```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```

1085 local function ulitem(s)
1086     return {"\\markdownRendererUlItem ",s,
1087           "\\markdownRendererUlItemEnd "}
1088 end
1089
1090 function self.bulletlist(items,tight)
1091     local buffer = {}
1092     for _,item in ipairs(items) do
1093         buffer[#buffer + 1] = ulitem(item)
1094     end
1095     local contents = util.intersperse(buffer,"\\n")
1096     if tight and options.tightLists then
1097         return {"\\markdownRendererUlBeginTight\\n",contents,
1098               "\\n\\markdownRendererUlEndTight "}
1099     else
1100         return {"\\markdownRendererUlBegin\\n",contents,
1101               "\\n\\markdownRendererUlEnd "}
1102     end
1103 end

```

Define `writer->ollist` as a function that will transform an input ordered list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not. If the optional parameter `startnum` is present, it should be used as the number of the first list item.

```

1104 local function olitem(s,num)
1105     if num ~= nil then
1106         return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
1107               "\\markdownRendererOlItemEnd "}
1108     else
1109         return {"\\markdownRendererOlItem ",s,

```

```

1110         "\\markdownRendererOlItemEnd "}
1111     end
1112 end
1113
1114 function self.orderedlist(items,tight,startnum)
1115     local buffer = {}
1116     local num = startnum
1117     for _,item in ipairs(items) do
1118         buffer[#buffer + 1] = olitem(item,num)
1119         if num ~= nil then
1120             num = num + 1
1121         end
1122     end
1123     local contents = util.intersperse(buffer,"\n")
1124     if tight and options.tightLists then
1125         return {"\\markdownRendererOlBeginTight\n",contents,
1126             "\n\\markdownRendererOlEndTight "}
1127     else
1128         return {"\\markdownRendererOlBegin\n",contents,
1129             "\n\\markdownRendererOlEnd "}
1130     end
1131 end

```

Define `writer->inline_html` and `writer->display_html` as functions that will transform an inline or block HTML element respectively to the output format, where `html` is the HTML input.

```

1132 function self.inline_html(html) return "" end
1133 function self.display_html(html) return "" end

```

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```

1134 local function dlitem(term, defs)
1135     local retVal = {"\\markdownRendererDlItem{",term,""}
1136     for _, def in ipairs(defs) do
1137         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
1138             "\\markdownRendererDlDefinitionEnd "}
1139     end
1140     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
1141     return retVal
1142 end
1143
1144 function self.definitionlist(items,tight)
1145     local buffer = {}
1146     for _,item in ipairs(items) do
1147         buffer[#buffer + 1] = dlitem(item.term, item.definitions)

```

```

1148     end
1149     if tight and options.tightLists then
1150         return {"\\markdownRendererDlBeginTight\n", buffer,
1151             "\n\\markdownRendererDlEndTight"}
1152     else
1153         return {"\\markdownRendererDlBegin\n", buffer,
1154             "\n\\markdownRendererDlEnd"}
1155     end
1156 end

```

Define `writer->emphasis` as a function that will transform an emphasized span `s` of input text to the output format.

```

1157 function self.emphasis(s)
1158     return {"\\markdownRendererEmphasis{" ,s,"}"}
1159 end

```

Define `writer->strong` as a function that will transform a strongly emphasized span `s` of input text to the output format.

```

1160 function self.strong(s)
1161     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
1162 end

```

Define `writer->blockquote` as a function that will transform an input block quote `s` to the output format.

```

1163 function self.blockquote(s)
1164     return {"\\markdownRendererBlockQuoteBegin\n",s,
1165         "\n\\markdownRendererBlockQuoteEnd "}
1166 end

```

Define `writer->verbatim` as a function that will transform an input code block `s` to the output format.

```

1167 function self.verbatim(s)
1168     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
1169     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
1170 end

```

Define `writer->codeFence` as a function that will transform an input fenced code block `s` with the infostring `i` to the output format.

```

1171 function self.fencedCode(i, s)
1172     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
1173     return {"\\markdownRendererInputFencedCode{" ,name,"}{",i,"}"}
1174 end

```

Define `writer->heading` as a function that will transform an input heading `s` at level `level` to the output format.

```

1175 function self.heading(s,level)
1176     local cmd
1177     if level == 1 then
1178         cmd = "\\markdownRendererHeadingOne"

```

```

1179     elsif level == 2 then
1180       cmd = "\\markdownRendererHeadingTwo"
1181     elsif level == 3 then
1182       cmd = "\\markdownRendererHeadingThree"
1183     elsif level == 4 then
1184       cmd = "\\markdownRendererHeadingFour"
1185     elsif level == 5 then
1186       cmd = "\\markdownRendererHeadingFive"
1187     elsif level == 6 then
1188       cmd = "\\markdownRendererHeadingSix"
1189     else
1190       cmd = ""
1191     end
1192     return {cmd,"{",s,""}
1193   end

```

Define `writer->note` as a function that will transform an input footnote `s` to the output format.

```

1194   function self.note(s)
1195     return {"\\markdownRendererFootnote{",s,""}
1196   end

```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is `true`, the citations should be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```

1197   function self.citations(text_cites, cites)
1198     local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
1199       "{", #cites, ""}
1200     for _,cite in ipairs(cites) do
1201       buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
1202         cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, ""}
1203     end
1204     return buffer
1205   end
1206

```

```

1207     return self
1208 end

```

3.1.4 Parsers

The `parsers` hash table stores PEG patterns that are static and can be reused between different `reader` objects.

```

1209 local parsers = {}

```

3.1.4.1 Basic Parsers

```

1210 parsers.percent = P("%")
1211 parsers.at = P("@")
1212 parsers.comma = P(",")
1213 parsers.asterisk = P("*")
1214 parsers.dash = P("-")
1215 parsers.plus = P("+")
1216 parsers.underscore = P("_")
1217 parsers.period = P(".")
1218 parsers.hash = P("#")
1219 parsers.ampersand = P("&")
1220 parsers.backtick = P("`")
1221 parsers.less = P("<")
1222 parsers.more = P(">")
1223 parsers.space = P(" ")
1224 parsers.squote = P("'")
1225 parsers.dquote = P('"')
1226 parsers.lparent = P("(")
1227 parsers.rparent = P(")")
1228 parsers.lbracket = P("[")
1229 parsers.rbracket = P("]")
1230 parsers.circumflex = P("^")
1231 parsers.slash = P("/")
1232 parsers.equal = P("=")
1233 parsers.colon = P(":")
1234 parsers.semicolon = P(";")
1235 parsers.exclamation = P("!")
1236 parsers.tilde = P("~")
1237 parsers.tab = P("\t")
1238 parsers.newline = P("\n")
1239 parsers.tightblocksep = P("\001")
1240
1241 parsers.digit = R("09")
1242 parsers.hexdigit = R("09", "af", "AF")
1243 parsers.letter = R("AZ", "az")
1244 parsers.alphanumeric = R("AZ", "az", "09")
1245 parsers.keyword = parsers.letter

```

```

1246                                     * parsers.alphanumeric^0
1247 parsers.citation_chars              = parsers.alphanumeric
1248                                     + S("#$%&-+<>~/_")
1249 parsers.internal_punctuation         = S(":,;,.?")
1250
1251 parsers.doubleasterisks               = P("**")
1252 parsers.doubleunderscores             = P("__")
1253 parsers.fourspace                    = P("    ")
1254
1255 parsers.any                           = P(1)
1256 parsers.fail                          = parsers.any - 1
1257
1258 parsers.escapable                     = S("\\'*_{}[]()+_ .!<>#-~:~^@;")
1259 parsers.anyescaped                    = P("\\") / "" * parsers.escapable
1260                                     + parsers.any
1261
1262 parsers.spacechar                      = S("\t ")
1263 parsers.spacing                       = S(" \n\r\t")
1264 parsers.nonpacechar                   = parsers.any - parsers.spacing
1265 parsers.optionalspace                 = parsers.spacechar^0
1266
1267 parsers.specialchar                   = S("*_ '&[]<!\\"
1268
1269 parsers.normalchar                    = parsers.any - (parsers.specialchar
1270                                                         + parsers.spacing
1271                                                         + parsers.tightblocksep)
1272 parsers.eof                           = -parsers.any
1273 parsers.nonindentpace                 = parsers.space^~3 * - parsers.spacechar
1274 parsers.indent                        = parsers.space^~3 * parsers.tab
1275                                     + parsers.fourspace / ""
1276 parsers.linechar                       = P(1 - parsers.newline)
1277
1278 parsers.blankline                     = parsers.optionalspace
1279                                     * parsers.newline / "\n"
1280 parsers.blanklines                    = parsers.blankline^0
1281 parsers.skipblanklines                 = (parsers.optionalspace * parsers.newline)^0
1282 parsers.indentedline                  = parsers.indent / ""
1283                                     * C(parsers.linechar^1 * parsers.newline^~1)
1284 parsers.optionallyindentedline         = parsers.indent^~1 / ""
1285                                     * C(parsers.linechar^1 * parsers.newline^~1)
1286 parsers.sp                             = parsers.spacing^0
1287 parsers.spnl                           = parsers.optionalspace
1288                                     * (parsers.newline * parsers.optionalspace)^~1
1289 parsers.line                           = parsers.linechar^0 * parsers.newline
1290                                     + parsers.linechar^1 * parsers.eof
1291 parsers.nonemptyline                   = parsers.line - parsers.blankline
1292

```



```

1293 parsers.chunk                = parsers.line * (parsers.optionallyindentedline
1294                                     - parsers.blankline)^0
1295
1296 -- block followed by 0 or more optionally
1297 -- indented blocks with first line indented.
1298 parsers.indented_blocks = function(bl)
1299   return Cs( bl
1300             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
1301             * (parsers.blankline^1 + parsers.eof) )
1302 end

```

3.1.4.2 Parsers Used for Markdown Lists

```

1303 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
1304
1305 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
1306                  * (parsers.tab + parsers.space^-3)
1307                  + parsers.space * parsers.bulletchar * #parsers.spacing
1308                  * (parsers.tab + parsers.space^-2)
1309                  + parsers.space * parsers.space * parsers.bulletchar
1310                  * #parsers.spacing
1311                  * (parsers.tab + parsers.space^-1)
1312                  + parsers.space * parsers.space * parsers.space
1313                  * parsers.bulletchar * #parsers.spacing
1314                  )

```

3.1.4.3 Parsers Used for Markdown Code Spans

```

1315 parsers.openticks    = Cg(parsers.backtick^1, "ticks")
1316
1317 local function captures_equal_length(s,i,a,b)
1318   return #a == #b and i
1319 end
1320
1321 parsers.closeticks    = parsers.space^-1
1322                      * Cmt(C(parsers.backtick^1)
1323                          * Cb("ticks"), captures_equal_length)
1324
1325 parsers.intickschar = (parsers.any - S("\n\r'"))
1326                      + (parsers.newline * -parsers.blankline)
1327                      + (parsers.space - parsers.closeticks)
1328                      + (parsers.backtick^1 - parsers.closeticks)
1329
1330 parsers.inticks       = parsers.openticks * parsers.space^-1
1331                      * C(parsers.intickschar^0) * parsers.closeticks

```

3.1.4.4 Parsers Used for Fenced Code Blocks

```

1332 local function captures_geq_length(s,i,a,b)
1333     return #a >= #b and i
1334 end
1335
1336 parsers.infostring      = (parsers.linechar - (parsers.backtick
1337     + parsers.space^1 * (parsers.newline + parsers.eof)))^0
1338
1339 local fenceindent
1340 parsers.fencehead      = function(char)
1341     return              C(parsers.nonindentspace) / function(s) fenceindent = #s end
1342     * Cg(char^3, "fencelength")
1343     * parsers.optionalspace * C(parsers.infostring)
1344     * parsers.optionalspace * (parsers.newline + parsers.eof)
1345 end
1346
1347 parsers.fencetail      = function(char)
1348     return              parsers.nonindentspace
1349     * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
1350     * parsers.optionalspace * (parsers.newline + parsers.eof)
1351     + parsers.eof
1352 end
1353
1354 parsers.fencedline      = function(char)
1355     return              C(parsers.line - parsers.fencetail(char))
1356     / function(s)
1357         i = 1
1358         remaining = fenceindent
1359         while true do
1360             c = s:sub(i, i)
1361             if c == " " and remaining > 0 then
1362                 remaining = remaining - 1
1363                 i = i + 1
1364             elseif c == "\t" and remaining > 3 then
1365                 remaining = remaining - 4
1366                 i = i + 1
1367             else
1368                 break
1369             end
1370         end
1371         return s:sub(i)
1372     end
1373 end

```

3.1.4.5 Parsers Used for Markdown Tags and Links

```

1374 parsers.leader          = parsers.space^-3
1375

```

```

1376 -- content in balanced brackets, parentheses, or quotes:
1377 parsers.bracketed = P{ parsers.lbracket
1378                       * ((parsers.anyescaped - (parsers.lbracket
1379                                                    + parsers.rbracket
1380                                                    + parsers.blankline^2)
1381                          ) + V(1))^0
1382                       * parsers.rbracket }
1383
1384 parsers.inparens    = P{ parsers.lparent
1385                          * ((parsers.anyescaped - (parsers.lparent
1386                                                    + parsers.rparent
1387                                                    + parsers.blankline^2)
1388                          ) + V(1))^0
1389                          * parsers.rparent }
1390
1391 parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
1392                          * ((parsers.anyescaped - (parsers.squote
1393                                                    + parsers.blankline^2)
1394                          ) + V(1))^0
1395                          * parsers.squote }
1396
1397 parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
1398                          * ((parsers.anyescaped - (parsers.dquote
1399                                                    + parsers.blankline^2)
1400                          ) + V(1))^0
1401                          * parsers.dquote }
1402
1403 -- bracketed tag for markdown links, allowing nested brackets:
1404 parsers.tag         = parsers.lbracket
1405                       * Cs((parsers.alphanumeric^1
1406                             + parsers.bracketed
1407                             + parsers.inticks
1408                             + (parsers.anyescaped
1409                               - (parsers.rbracket + parsers.blankline^2)))^0)
1410                       * parsers.rbracket
1411
1412 -- url for markdown links, allowing nested brackets:
1413 parsers.url         = parsers.less * Cs((parsers.anyescaped
1414                                           - parsers.more)^0)
1415                       * parsers.more
1416                       + Cs((parsers.inparens + (parsers.anyescaped
1417                                           - parsers.spacing
1418                                           - parsers.rparent))^1)
1419
1420 -- quoted text, possibly with nested quotes:
1421 parsers.title_s     = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
1422                                           + parsers.squoted)^0)

```

```

1423                                     * parsers.squote
1424
1425 parsers.title_d      = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
1426                                     + parsers.dquoted)^0)
1427                                     * parsers.dquote
1428
1429 parsers.title_p      = parsers.lparent
1430                       * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
1431                       * parsers.rparent
1432
1433 parsers.title        = parsers.title_d + parsers.title_s + parsers.title_p
1434
1435 parsers.optionaltitle
1436                       = parsers.spnl * parsers.title * parsers.spacechar^0
1437                       + Cc("")

```

3.1.4.6 Parsers Used for iA Writer Content Blocks

```

1438 parsers.contentblock_tail
1439                       = parsers.optionaltitle
1440                       * (parsers.newline + parsers.eof)
1441
1442 -- case insensitive online image suffix:
1443 parsers.onlineimagesuffix
1444                       = (function(...)
1445                           local parser = nil
1446                           for _,suffix in ipairs({...}) do
1447                               local pattern=nil
1448                               for i=1,#suffix do
1449                                   local char=suffix:sub(i,i)
1450                                   char = S(char:lower()..char:upper())
1451                                   if pattern == nil then
1452                                       pattern = char
1453                                   else
1454                                       pattern = pattern * char
1455                                   end
1456                               end
1457                               if parser == nil then
1458                                   parser = pattern
1459                               else
1460                                   parser = parser + pattern
1461                               end
1462                           end
1463                           return parser
1464                       end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
1465
1466 -- online image url for iA Writer content blocks with mandatory suffix,

```

```

1467 -- allowing nested brackets:
1468 parsers.onlineimageurl
1469         = (parsers.less
1470           * Cs((parsers.anyescaped
1471               - parsers.more
1472               - #(parsers.period
1473                   * parsers.onlineimagesuffix
1474                   * parsers.more
1475                   * parsers.contentblock_tail))^0)
1476           * parsers.period
1477           * Cs(parsers.onlineimagesuffix)
1478           * parsers.more
1479           + (Cs((parsers.inparens
1480               + (parsers.anyescaped
1481                   - parsers.spacing
1482                   - parsers.rparent
1483                   - #(parsers.period
1484                       * parsers.onlineimagesuffix
1485                       * parsers.contentblock_tail))))^0)
1486           * parsers.period
1487           * Cs(parsers.onlineimagesuffix))
1488           ) * Cc("onlineimage")
1489
1490 -- filename for iA Writer content blocks with mandatory suffix:
1491 parsers.localfilepath
1492         = parsers.slash
1493         * Cs((parsers.anyescaped
1494             - parsers.tab
1495             - parsers.newline
1496             - #(parsers.period
1497                 * parsers.alphanumeric^1
1498                 * parsers.contentblock_tail))^1)
1499         * parsers.period
1500         * Cs(parsers.alphanumeric^1)
1501         * Cc("localfile")

```

3.1.4.7 Parsers Used for Citations

```

1502 parsers.citation_name = Cs(parsers.dash^-1) * parsers.at
1503         * Cs(parsers.citation_chars
1504             * (((parsers.citation_chars + parsers.internal_punctuation
1505                 - parsers.comma - parsers.semicolon)
1506                 * -#((parsers.internal_punctuation - parsers.comma
1507                     - parsers.semicolon)^0
1508                     * -(parsers.citation_chars + parsers.internal_punctuat.
1509                         - parsers.comma - parsers.semicolon)))^0
1510                 * parsers.citation_chars)^-1)

```

```

1511
1512 parsers.citation_body_prenote
1513         = Cs((parsers.alphanumeric^1
1514             + parsers.bracketed
1515             + parsers.inticks
1516             + (parsers.anyescaped
1517               - (parsers.rbracket + parsers.blankline^2))
1518             - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
1519
1520 parsers.citation_body_postnote
1521         = Cs((parsers.alphanumeric^1
1522             + parsers.bracketed
1523             + parsers.inticks
1524             + (parsers.anyescaped
1525               - (parsers.rbracket + parsers.semicolon
1526                 + parsers.blankline^2))
1527             - (parsers.spnl * parsers.rbracket))^0)
1528
1529 parsers.citation_body_chunk
1530         = parsers.citation_body_prenote
1531         * parsers.spnl * parsers.citation_name
1532         * ((parsers.internal_punctuation - parsers.semicolon)
1533           * parsers.spnl)^-1
1534         * parsers.citation_body_postnote
1535
1536 parsers.citation_body
1537         = parsers.citation_body_chunk
1538         * (parsers.semicolon * parsers.spnl
1539           * parsers.citation_body_chunk)^0
1540
1541 parsers.citation_headless_body_postnote
1542         = Cs((parsers.alphanumeric^1
1543             + parsers.bracketed
1544             + parsers.inticks
1545             + (parsers.anyescaped
1546               - (parsers.rbracket + parsers.at
1547                 + parsers.semicolon + parsers.blankline^2))
1548             - (parsers.spnl * parsers.rbracket))^0)
1549
1550 parsers.citation_headless_body
1551         = parsers.citation_headless_body_postnote
1552         * (parsers.sp * parsers.semicolon * parsers.spnl
1553           * parsers.citation_body_chunk)^0

```

3.1.4.8 Parsers Used for Footnotes

```

1554 local function strip_first_char(s)

```

```

1555     return s:sub(2)
1556 end
1557
1558 parsers.RawNoteRef = #(parsers.lbracket * parsers.circumflex)
1559                      * parsers.tag / strip_first_char

```

3.1.4.9 Parsers Used for HTML

```

1560 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
1561 parsers.keyword_exact = function(s)
1562     local parser = P(0)
1563     for i=1,#s do
1564         local c = s:sub(i,i)
1565         local m = c .. upper(c)
1566         parser = parser * S(m)
1567     end
1568     return parser
1569 end
1570
1571 parsers.block_keyword =
1572     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
1573     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
1574     parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
1575     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
1576     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
1577     parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
1578     parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
1579     parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
1580     parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
1581     parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
1582     parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
1583     parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
1584     parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
1585     parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
1586     parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
1587     parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
1588     parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
1589     parsers.keyword_exact("td") + parsers.keyword_exact("tr")
1590
1591 -- There is no reason to support bad html, so we expect quoted attributes
1592 parsers.htmlattributevalue
1593     = parsers.squote * (parsers.any - (parsers.blankline
1594                                     + parsers.squote))^0
1595                       * parsers.squote
1596     + parsers.dquote * (parsers.any - (parsers.blankline
1597                                     + parsers.dquote))^0
1598                       * parsers.dquote

```

```

1599
1600 parsers.htmlattribute      = parsers.spacing^1
1601                             * (parsers.alphanumeric + S("_-"))^1
1602                             * parsers.sp * parsers.equal * parsers.sp
1603                             * parsers.htmlattributevalue
1604
1605 parsers.htmlcomment         = P("<!--") * (parsers.any - P("-->"))^0 * P("-->")
1606
1607 parsers.htmlinstruction      = P("<?") * (parsers.any - P("?>" ))^0 * P("?>" )
1608
1609 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
1610                   * parsers.sp * parsers.more
1611
1612 parsers.openelt_exact = function(s)
1613   return parsers.less * parsers.sp * parsers.keyword_exact(s)
1614         * parsers.htmlattribute^0 * parsers.sp * parsers.more
1615 end
1616
1617 parsers.openelt_block = parsers.sp * parsers.block_keyword
1618                   * parsers.htmlattribute^0 * parsers.sp * parsers.more
1619
1620 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
1621                   * parsers.keyword * parsers.sp * parsers.more
1622
1623 parsers.closeelt_exact = function(s)
1624   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
1625         * parsers.sp * parsers.more
1626 end
1627
1628 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
1629                   * parsers.htmlattribute^0 * parsers.sp * parsers.slash
1630                   * parsers.more
1631
1632 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
1633                   * parsers.htmlattribute^0 * parsers.sp * parsers.slash
1634                   * parsers.more
1635
1636 parsers.displaytext = (parsers.any - parsers.less)^1
1637
1638 -- return content between two matched HTML tags
1639 parsers.in_matched = function(s)
1640   return { parsers.openelt_exact(s)
1641         * (V(1) + parsers.displaytext
1642         + (parsers.less - parsers.closeelt_exact(s)))^0
1643         * parsers.closeelt_exact(s) }
1644 end
1645

```



```

1646 local function parse_matched_tags(s,pos)
1647   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
1648   return lpeg.match(parsers.in_matched(t),s,pos-1)
1649 end
1650
1651 parsers.in_matched_block_tags = parsers.less
1652                               * Cmt(#parsers.openelt_block, parse_matched_tags)
1653
1654 parsers.displayhtml = parsers.htmlcomment
1655                     + parsers.emptyelt_block
1656                     + parsers.openelt_exact("hr")
1657                     + parsers.in_matched_block_tags
1658                     + parsers.htmlinstruction
1659
1660 parsers.inlinehtml = parsers.emptyelt_any
1661                   + parsers.htmlcomment
1662                   + parsers.htmlinstruction
1663                   + parsers.openelt_any
1664                   + parsers.closeelt_any

```

3.1.4.10 Parsers Used for HTML entities

```

1665 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
1666                   * C(parsers.hexdigit^1) * parsers.semicolon
1667 parsers.decentity = parsers.ampersand * parsers.hash
1668                   * C(parsers.digit^1) * parsers.semicolon
1669 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
1670                   * parsers.semicolon

```

3.1.4.11 Helpers for References

```

1671 -- parse a reference definition: [foo]: /bar "title"
1672 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
1673                               * parsers.spacechar^0 * parsers.url
1674                               * parsers.optionaltitle * parsers.blankline^1

```

3.1.4.12 Inline Elements

```

1675 parsers.Inline      = V("Inline")
1676
1677 -- parse many p between starter and ender
1678 parsers.between = function(p, starter, ender)
1679   local ender2 = B(parsers.nonspacechar) * ender
1680   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
1681 end
1682
1683 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

3.1.4.13 Block Elements

```
1684 parsers.Block          = V("Block")
1685
1686 parsers.OnlineImageURL
1687     = parsers.leader
1688     * parsers.onlineimageurl
1689     * parsers.optionaltitle
1690
1691 parsers.LocalFilePath
1692     = parsers.leader
1693     * parsers.localfilepath
1694     * parsers.optionaltitle
1695
1696 parsers.TildeFencedCode
1697     = parsers.fencehead(parsers.tilde)
1698     * Cs(parsers.fencedline(parsers.tilde)^0)
1699     * parsers.fencetail(parsers.tilde)
1700
1701 parsers.BacktickFencedCode
1702     = parsers.fencehead(parsers.backtick)
1703     * Cs(parsers.fencedline(parsers.backtick)^0)
1704     * parsers.fencetail(parsers.backtick)
1705
1706 parsers.lineof = function(c)
1707     return (parsers.leader * (P(c) * parsers.optionalspace)^3
1708           * (parsers.newline * parsers.blankline^1
1709           + parsers.newline^-1 * parsers.eof))
1710 end
```

3.1.4.14 Lists

```
1711 parsers.defstartchar = S("~:")
1712 parsers.defstart      = ( parsers.defstartchar * #parsers.spacing
1713                          * (parsers.tab + parsers.space^-3)
1714      + parsers.space * parsers.defstartchar * #parsers.spacing
1715                          * (parsers.tab + parsers.space^-2)
1716      + parsers.space * parsers.space * parsers.defstartchar
1717                          * #parsers.spacing
1718                          * (parsers.tab + parsers.space^-1)
1719      + parsers.space * parsers.space * parsers.space
1720                          * parsers.defstartchar * #parsers.spacing
1721      )
1722
1723 parsers.dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
```

3.1.4.15 Headings

```

1724 -- parse Atx heading start and return level
1725 parsers.HeadingStart = #parsers.hash * C(parsers.hash~-6)
1726                       * -parsers.hash / length
1727
1728 -- parse setext header ending and return level
1729 parsers.HeadingLevel = parsers.equal~1 * Cc(1) + parsers.dash~1 * Cc(2)
1730
1731 local function strip_atx_end(s)
1732   return s:gsub("#%s]*\n$", "")
1733 end

```

3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `reader` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `reader.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

The `reader.new` method creates and returns a new T_EX reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```

1734 M.reader = {}
1735 function M.reader.new(writer, options)
1736   local self = {}
1737   options = options or {}

```

Make the `options` table inherit from the `defaultOptions` table.

```

1738   setmetatable(options, { __index = function (_, key)
1739     return defaultOptions[key] end })

```

3.1.5.1 Top-Level Helper Functions Define `normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```

1740   local function normalize_tag(tag)
1741     return unicode.utf8.lower(
1742       gsub(util.rope_to_string(tag), "[\n\r\t]+", " "))
1743   end

```

Define `expandtabs` either as an identity function, when the `preserveTabs` Lua inrface option is `true`, or to a function that expands tabs into spaces otherwise.

```
1744 local expandtabs
1745 if options.preserveTabs then
1746     expandtabs = function(s) return s end
1747 else
1748     expandtabs = function(s)
1749         if s:find("\t") then
1750             return s:gsub("[^\n]*", util.expand_tabs_in_line)
1751         else
1752             return s
1753         end
1754     end
1755 end
```

The `larsers` (as in “local `parsers`”) hash table stores PEG patterns that depend on the received `options`, which impedes their reuse between different `reader` objects.

```
1756 local larsers = {}
```

3.1.5.2 Top-Level Parser Functions

```
1757 local function create_parser(name, grammar)
1758     return function(str)
1759         local res = lpeg.match(grammar(), str)
1760         if res == nil then
1761             error(format("%s failed on:\n%s", name, str:sub(1,20)))
1762         else
1763             return res
1764         end
1765     end
1766 end
1767
1768 local parse_blocks
1769     = create_parser("parse_blocks",
1770         function()
1771             return larsers.blocks
1772         end)
1773
1774 local parse_blocks_toplevel
1775     = create_parser("parse_blocks_toplevel",
1776         function()
1777             return larsers.blocks_toplevel
1778         end)
1779
1780 local parse_inlines
1781     = create_parser("parse_inlines",
1782         function()
```

```

1783         return larsers.inlines
1784     end)
1785
1786     local parse_inlines_no_link
1787     = create_parser("parse_inlines_no_link",
1788         function()
1789             return larsers.inlines_no_link
1790         end)
1791
1792     local parse_inlines_no_inline_note
1793     = create_parser("parse_inlines_no_inline_note",
1794         function()
1795             return larsers.inlines_no_inline_note
1796         end)
1797
1798     local parse_inlines_nbsp
1799     = create_parser("parse_inlines_nbsp",
1800         function()
1801             return larsers.inlines_nbsp
1802         end)

```

3.1.5.3 Parsers Used for Markdown Lists (local)

```

1803     if options.hashEnumerators then
1804         larsers.dig = parsers.digit + parsers.hash
1805     else
1806         larsers.dig = parsers.digit
1807     end
1808
1809     larsers.enumerator = C(larsers.dig^3 * parsers.period) * #parsers.spacing
1810     + C(larsers.dig^2 * parsers.period) * #parsers.spacing
1811     * (parsers.tab + parsers.space^1)
1812     + C(larsers.dig * parsers.period) * #parsers.spacing
1813     * (parsers.tab + parsers.space^2)
1814     + parsers.space * C(larsers.dig^2 * parsers.period)
1815     * #parsers.spacing
1816     + parsers.space * C(larsers.dig * parsers.period)
1817     * #parsers.spacing
1818     * (parsers.tab + parsers.space^1)
1819     + parsers.space * parsers.space * C(larsers.dig^1
1820     * parsers.period) * #parsers.spacing

```

3.1.5.4 Parsers Used for Blockquotes (local)

```

1821     -- strip off leading > and indents, and run through blocks
1822     larsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^1)/""
1823     * parsers.linechar^0 * parsers.newline)^1
1824     * (-(parsers.leader * parsers.more

```

```

1825             + parsers.blankline) * parsers.linechar^1
1826             * parsers.newline)^0
1827
1828 if not options.breakableBlockquotes then
1829     larsers.blockquote_body = larsers.blockquote_body
1830     * (parsers.blankline^0 / "")
1831 end

```

3.1.5.5 Parsers Used for Citations (local)

```

1832 larsers.citations = function(text_cites, raw_cites)
1833     local function normalize(str)
1834         if str == "" then
1835             str = nil
1836         else
1837             str = (options.citationNbsps and parse_inlines_nbsp or
1838                 parse_inlines)(str)
1839         end
1840         return str
1841     end
1842
1843     local cites = {}
1844     for i = 1,#raw_cites,4 do
1845         cites[#cites+1] = {
1846             prenote = normalize(raw_cites[i]),
1847             suppress_author = raw_cites[i+1] == "-",
1848             name = writer.citation(raw_cites[i+2]),
1849             postnote = normalize(raw_cites[i+3]),
1850         }
1851     end
1852     return writer.citations(text_cites, cites)
1853 end

```

3.1.5.6 Parsers Used for Footnotes (local)

```

1854 local rawnotes = {}
1855
1856 -- like indirect_link
1857 local function lookup_note(ref)
1858     return function()
1859         local found = rawnotes[normalize_tag(ref)]
1860         if found then
1861             return writer.note(parse_blocks_toplevel(found))
1862         else
1863             return {"[", parse_inlines("^" .. ref), "]" }
1864         end
1865     end
1866 end

```

```

1867
1868 local function register_note(ref,rawnote)
1869     rawnotes[normalize_tag(ref)] = rawnote
1870     return ""
1871 end
1872
1873 larsers.NoteRef      = parsers.RawNoteRef / lookup_note
1874
1875
1876 larsers.NoteBlock    = parsers.leader * parsers.RawNoteRef * parsers.colon
1877                       * parsers.spnl * parsers.indented_blocks(parsers.chunk)
1878                       / register_note
1879
1880 larsers.InlineNote   = parsers.circumflex
1881                       * (parsers.tag / parse_inlines_no_inline_note) -- no notes inside
1882                       / writer.note

```

3.1.5.7 Helpers for Links and References (local)

```

1883 -- List of references defined in the document
1884 local references
1885
1886 -- add a reference to the list
1887 local function register_link(tag,url,title)
1888     references[normalize_tag(tag)] = { url = url, title = title }
1889     return ""
1890 end
1891
1892 -- lookup link reference and return either
1893 -- the link or nil and fallback text.
1894 local function lookup_reference(label,sps,tag)
1895     local tagpart
1896     if not tag then
1897         tag = label
1898         tagpart = ""
1899     elseif tag == "" then
1900         tag = label
1901         tagpart = "[]"
1902     else
1903         tagpart = {"[", parse_inlines(tag), "]" }
1904     end
1905     if sps then
1906         tagpart = {sps, tagpart}
1907     end
1908     local r = references[normalize_tag(tag)]
1909     if r then
1910         return r

```

```

1911         else
1912             return nil, {"[", parse_inlines(label), "]", tagpart}
1913         end
1914     end
1915
1916     -- lookup link reference and return a link, if the reference is found,
1917     -- or a bracketed label otherwise.
1918     local function indirect_link(label,sps,tag)
1919         return function()
1920             local r,fallback = lookup_reference(label,sps,tag)
1921             if r then
1922                 return writer.link(parse_inlines_no_link(label), r.url, r.title)
1923             else
1924                 return fallback
1925             end
1926         end
1927     end
1928
1929     -- lookup image reference and return an image, if the reference is found,
1930     -- or a bracketed label otherwise.
1931     local function indirect_image(label,sps,tag)
1932         return function()
1933             local r,fallback = lookup_reference(label,sps,tag)
1934             if r then
1935                 return writer.image(writer.string(label), r.url, r.title)
1936             else
1937                 return {"!", fallback}
1938             end
1939         end
1940     end

```

3.1.5.8 Inline Elements (local)

```

1941     larsers.Str          = parsers.normalchar^1 / writer.string
1942
1943     larsers.Symbol       = (parsers.specialchar - parsers.tightblocksep)
1944                          / writer.string
1945
1946     larsers.Ellipsis     = P("...") / writer.ellipsis
1947
1948     larsers.Smart        = larsers.Ellipsis
1949
1950     larsers.Code         = parsers.inticks / writer.code
1951
1952     if options.blankBeforeBlockquote then
1953         larsers.bqstart = parsers.fail
1954     else

```



```

1955     larsers.bqstart = parsers.more
1956 end
1957
1958 if options.blankBeforeHeading then
1959     larsers.headerstart = parsers.fail
1960 else
1961     larsers.headerstart = parsers.hash
1962                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
1963                         * parsers.optionalspace * parsers.newline)
1964 end
1965
1966 if not options.fencedCode or options.blankBeforeCodeFence then
1967     larsers.fencestart = parsers.fail
1968 else
1969     larsers.fencestart = parsers.fencehead(parsers.backtick)
1970                       + parsers.fencehead(parsers.tilde)
1971 end
1972
1973 larsers.Endline    = parsers.newline * -( -- newline, but not before...
1974                    parsers.blankline -- paragraph break
1975                    + parsers.tightblocksep -- nested list
1976                    + parsers.eof          -- end of document
1977                    + larsers.bqstart
1978                    + larsers.headerstart
1979                    + larsers.fencestart
1980                    ) * parsers.spacechar^0 / writer.space
1981
1982 larsers.Space      = parsers.spacechar^2 * larsers.Endline / writer.linebreak
1983                    + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
1984                    + parsers.spacechar^1 * larsers.Endline^-1
1985                    * parsers.optionalspace / writer.space
1986
1987 larsers.NonbreakingEndline
1988     = parsers.newline * -( -- newline, but not before...
1989        parsers.blankline -- paragraph break
1990        + parsers.tightblocksep -- nested list
1991        + parsers.eof          -- end of document
1992        + larsers.bqstart
1993        + larsers.headerstart
1994        + larsers.fencestart
1995        ) * parsers.spacechar^0 / writer.nbsp
1996
1997 larsers.NonbreakingSpace
1998     = parsers.spacechar^2 * larsers.Endline / writer.linebreak
1999     + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
2000     + parsers.spacechar^1 * larsers.Endline^-1
2001     * parsers.optionalspace / writer.nbsp

```

```

2002
2003 if options.underscores then
2004     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
2005                                     parsers.doubleasterisks)
2006                     + parsers.between(parsers.Inline, parsers.doubleunderscores,
2007                                     parsers.doubleunderscores)
2008                     ) / writer.strong
2009
2010     larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
2011                                     parsers.asterisk)
2012                     + parsers.between(parsers.Inline, parsers.underscore,
2013                                     parsers.underscore)
2014                     ) / writer.emphasis
2015 else
2016     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
2017                                     parsers.doubleasterisks)
2018                     ) / writer.strong
2019
2020     larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
2021                                     parsers.asterisk)
2022                     ) / writer.emphasis
2023 end
2024
2025 larsers.AutoLinkUrl    = parsers.less
2026                       * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
2027                       * parsers.more
2028                       / function(url)
2029                           return writer.link(writer.string(url), url)
2030                       end
2031
2032 larsers.AutoLinkEmail = parsers.less
2033                       * C((parsers.alphanumeric + S("-._+"))^1
2034                       * P("@") * parsers.urlchar^1)
2035                       * parsers.more
2036                       / function(email)
2037                           return writer.link(writer.string(email),
2038                                               "mailto:.."email)
2039                       end
2040
2041 larsers.DirectLink     = (parsers.tag / parse_inlines_no_link) -- no links inside link
2042                       * parsers.spnl
2043                       * parsers.lparent
2044                       * (parsers.url + C("")) -- link can be empty [foo]()
2045                       * parsers.optionaltitle
2046                       * parsers.rparent
2047                       / writer.link
2048

```

```

2049 larsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-1
2050                        / indirect_link
2051
2052 -- parse a link or image (direct or indirect)
2053 larsers.Link          = larsers.DirectLink + larsers.IndirectLink
2054
2055 larsers.DirectImage   = parsers.exclamation
2056                        * (parsers.tag / parse_inlines)
2057                        * parsers.spnl
2058                        * parsers.lparent
2059                        * (parsers.url + Cc("")) -- link can be empty [foo]()
2060                        * parsers.optionaltitle
2061                        * parsers.rparent
2062                        / writer.image
2063
2064 larsers.IndirectImage = parsers.exclamation * parsers.tag
2065                        * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
2066
2067 larsers.Image         = larsers.DirectImage + larsers.IndirectImage
2068
2069 larsers.TextCitations = Ct(Cc(""))
2070                        * parsers.citation_name
2071                        * ((parsers.spnl
2072                           * parsers.lbracket
2073                           * parsers.citation_headless_body
2074                           * parsers.rbracket) + Cc("")))
2075                        / function(raw_cites)
2076                           return larsers.citations(true, raw_cites)
2077                        end
2078
2079 larsers.ParenthesizedCitations
2080                        = Ct(parsers.lbracket
2081                           * parsers.citation_body
2082                           * parsers.rbracket)
2083                        / function(raw_cites)
2084                           return larsers.citations(false, raw_cites)
2085                        end
2086
2087 larsers.Citations     = larsers.TextCitations + larsers.ParenthesizedCitations
2088
2089 -- avoid parsing long strings of * or _ as emph/strong
2090 larsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
2091                        / writer.string
2092
2093 larsers.EscapedChar    = S("\\") * C(parsers.escapable) / writer.string
2094
2095 larsers.InlineHtml     = C(parsers.inlinehtml) / writer.inline_html

```

```

2096
2097 larsers.HtmlEntity      = parsers.hexentity / entities.hex_entity / writer.string
2098                        + parsers.decentity / entities.dec_entity / writer.string
2099                        + parsers.tagentity / entities.char_entity / writer.string

```

3.1.5.9 Block Elements (local)

```

2100 larsers.ContentBlock = parsers.leader
2101                        * (parsers.localfilepath + parsers.onlineimageurl)
2102                        * parsers.contentblock_tail
2103                        / writer.contentblock
2104
2105 larsers.DisplayHtml   = C(parsers.displayhtml)
2106                        / expandtabs / writer.display_html
2107
2108 larsers.Verbatim      = Cs( (parsers.blanklines
2109                            * ((parsers.indentedline - parsers.blankline))^1)^1
2110                            ) / expandtabs / writer.verbatim
2111
2112 larsers.FencedCode    = (parsers.TildeFencedCode
2113                        + parsers.BacktickFencedCode)
2114                        / function(infostring, code)
2115                            return writer.fencedCode(writer.string(infostring),
2116                                                        expandtabs(code))
2117                        end
2118
2119 larsers.Blockquote     = Cs(larsers.blockquote_body^1)
2120                        / parse_blocks_toplevel / writer.blockquote
2121
2122 larsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
2123                        + parsers.lineof(parsers.dash)
2124                        + parsers.lineof(parsers.underscore)
2125                        ) / writer.hrule
2126
2127 larsers.Reference      = parsers.define_reference_parser / register_link
2128
2129 larsers.Paragraph      = parsers.nonindentSPACE * Ct(parsers.Inline^1)
2130                        * parsers.newline
2131                        * ( parsers.blankline^1
2132                        + #parsers.hash
2133                        + #(parsers.leader * parsers.more * parsers.space^-1)
2134                        )
2135                        / writer.paragraph
2136
2137 larsers.ToplevelParagraph
2138                        = parsers.nonindentSPACE * Ct(parsers.Inline^1)
2139                        * ( parsers.newline

```

```

2140             * ( parsers.blankline^1
2141               + #parsers.hash
2142               + #(parsers.leader * parsers.more * parsers.space^-1)
2143               + parsers.eof
2144             )
2145             + parsers.eof )
2146             / writer.paragraph
2147
2148     larsers.Plain      = parsers.nonindentSPACE * Ct(parsers.Inline^1)
2149                     / writer.plain

```

3.1.5.10 Lists (local)

```

2150     larsers.starter = parsers.bullet + larsers.enumerator
2151
2152     -- we use \001 as a separator between a tight list item and a
2153     -- nested list under it.
2154     larsers.NestedList      = Cs((parsers.optionallyindentedline
2155                               - larsers.starter)^1)
2156                               / function(a) return "\001"..a end
2157
2158     larsers.ListBlockLine   = parsers.optionallyindentedline
2159                               - parsers.blankline - (parsers.indent^-1
2160                                                       * larsers.starter)
2161
2162     larsers.ListBlock       = parsers.line * larsers.ListBlockLine^0
2163
2164     larsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
2165                               * larsers.ListBlock
2166
2167     larsers.TightListItem = function(starter)
2168         return -larsers.HorizontalRule
2169               * (Cs(starter / "" * larsers.ListBlock * larsers.NestedList^-1)
2170                 / parse_blocks)
2171               * -(parsers.blanklines * parsers.indent)
2172     end
2173
2174     larsers.LooseListItem = function(starter)
2175         return -larsers.HorizontalRule
2176               * Cs( starter / "" * larsers.ListBlock * Cc("\n")
2177                   * (larsers.NestedList + larsers.ListContinuationBlock^0)
2178                   * (parsers.blanklines / "\n\n")
2179                   ) / parse_blocks
2180     end
2181
2182     larsers.BulletList = ( Ct(larsers.TightListItem(parsers.bullet)^1) * Cc(true)
2183                           * parsers.skipblanklines * -parsers.bullet

```

```

2184             + Ct(larsers.LooseListItem(parsers.bullet)^1) * Cc(false)
2185             * parsers.skipblanklines )
2186         / writer.bulletlist
2187
2188     local function ordered_list(items,tight,startNumber)
2189         if options.startNumber then
2190             startNumber = tonumber(startNumber) or 1 -- fallback for '#'
2191         else
2192             startNumber = nil
2193         end
2194         return writer.orderedlist(items,tight,startNumber)
2195     end
2196
2197     larsers.OrderedList = Cg(larsers.enumerator, "listtype") *
2198         ( Ct(larsers.TightListItem(Cb("listtype")))
2199           * larsers.TightListItem(larsers.enumerator)^0)
2200       * Cc(true) * parsers.skipblanklines * -larsers.enumerator
2201       + Ct(larsers.LooseListItem(Cb("listtype")))
2202         * larsers.LooseListItem(larsers.enumerator)^0)
2203       * Cc(false) * parsers.skipblanklines
2204       ) * Cb("listtype") / ordered_list
2205
2206     local function definition_list_item(term, defs, tight)
2207         return { term = parse_inlines(term), definitions = defs }
2208     end
2209
2210     larsers.DefinitionListItemLoose = C(parsers.line) * parsers.skipblanklines
2211         * Ct((parsers.defstart
2212             * parsers.indented_blocks(parsers.dlchunk)
2213             / parse_blocks_toplevel)^1)
2214         * Cc(false) / definition_list_item
2215
2216     larsers.DefinitionListItemTight = C(parsers.line)
2217         * Ct((parsers.defstart * parsers.dlchunk
2218             / parse_blocks)^1)
2219         * Cc(true) / definition_list_item
2220
2221     larsers.DefinitionList = ( Ct(larsers.DefinitionListItemLoose^1) * Cc(false)
2222       + Ct(larsers.DefinitionListItemTight^1)
2223       * (parsers.skipblanklines
2224         * -larsers.DefinitionListItemLoose * Cc(true))
2225       ) / writer.definitionlist

```

3.1.5.11 Blank (local)

```

2226     larsers.Blank           = parsers.blankline / ""
2227
2227     + larsers.NoteBlock

```

```

2228         + larsers.Reference
2229         + (parsers.tightblocksep / "\n")

```

3.1.5.12 Headings (local)

```

2230 -- parse atx header
2231 larsers.AtHeading = Cg(parsers.HeadingStart,"level")
2232                 * parsers.optionalspace
2233                 * (C(parsers.line) / strip_atx_end / parse_inlines)
2234                 * Cb("level")
2235                 / writer.heading
2236
2237 -- parse setext header
2238 larsers.SetextHeading = #(parsers.line * S("--"))
2239                       * Ct(parsers.line / parse_inlines)
2240                       * parsers.HeadingLevel
2241                       * parsers.optionalspace * parsers.newline
2242                       / writer.heading
2243
2244 larsers.Heading = larsers.AtHeading + larsers.SetextHeading

```

3.1.5.13 Syntax Specification

```

2245 local syntax =
2246   { "Blocks",
2247
2248     Blocks      = larsers.Blank^0 * parsers.Block^-1
2249                  * (larsers.Blank^0 / function()
2250                      return writer.interblocksep
2251                      end
2252                      * parsers.Block)^0
2253                  * larsers.Blank^0 * parsers.eof,
2254
2255     Blank       = larsers.Blank,
2256
2257     Block       = V("ContentBlock")
2258                  + V("Blockquote")
2259                  + V("Verbatim")
2260                  + V("FencedCode")
2261                  + V("HorizontalRule")
2262                  + V("BulletList")
2263                  + V("OrderedList")
2264                  + V("Heading")
2265                  + V("DefinitionList")
2266                  + V("DisplayHtml")
2267                  + V("Paragraph")
2268                  + V("Plain"),
2269

```

2270	ContentBlock	= larsers.ContentBlock,
2271	Blockquote	= larsers.Blockquote,
2272	Verbatim	= larsers.Verbatim,
2273	FencedCode	= larsers.FencedCode,
2274	HorizontalRule	= larsers.HorizontalRule,
2275	BulletList	= larsers.BulletList,
2276	OrderedList	= larsers.OrderedList,
2277	Heading	= larsers.Heading,
2278	DefinitionList	= larsers.DefinitionList,
2279	DisplayHtml	= larsers.DisplayHtml,
2280	Paragraph	= larsers.Paragraph,
2281	Plain	= larsers.Plain,
2282		
2283	Inline	= V("Str")
2284		+ V("Space")
2285		+ V("Endline")
2286		+ V("U1OrStarLine")
2287		+ V("Strong")
2288		+ V("Emph")
2289		+ V("InlineNote")
2290		+ V("NoteRef")
2291		+ V("Citations")
2292		+ V("Link")
2293		+ V("Image")
2294		+ V("Code")
2295		+ V("AutoLinkUrl")
2296		+ V("AutoLinkEmail")
2297		+ V("InlineHtml")
2298		+ V("HtmlEntity")
2299		+ V("EscapedChar")
2300		+ V("Smart")
2301		+ V("Symbol"),
2302		
2303	Str	= larsers.Str,
2304	Space	= larsers.Space,
2305	Endline	= larsers.Endline,
2306	U1OrStarLine	= larsers.U1OrStarLine,
2307	Strong	= larsers.Strong,
2308	Emph	= larsers.Emph,
2309	InlineNote	= larsers.InlineNote,
2310	NoteRef	= larsers.NoteRef,
2311	Citations	= larsers.Citations,
2312	Link	= larsers.Link,
2313	Image	= larsers.Image,
2314	Code	= larsers.Code,
2315	AutoLinkUrl	= larsers.AutoLinkUrl,
2316	AutoLinkEmail	= larsers.AutoLinkEmail,


```

2317     InlineHtml          = larsers.InlineHtml,
2318     HtmlEntity           = larsers.HtmlEntity,
2319     EscapedChar          = larsers.EscapedChar,
2320     Smart                 = larsers.Smart,
2321     Symbol                = larsers.Symbol,
2322 }
2323
2324 if not options.citations then
2325     syntax.Citations = parsers.fail
2326 end
2327
2328 if not options.contentBlocks then
2329     syntax.ContentBlock = parsers.fail
2330 end
2331
2332 if not options.codeSpans then
2333     syntax.Code = parsers.fail
2334 end
2335
2336 if not options.definitionLists then
2337     syntax.DefinitionList = parsers.fail
2338 end
2339
2340 if not options.fencedCode then
2341     syntax.FencedCode = parsers.fail
2342 end
2343
2344 if not options.footnotes then
2345     syntax.NoteRef = parsers.fail
2346 end
2347
2348 if not options.html then
2349     syntax.DisplayHtml = parsers.fail
2350     syntax.InlineHtml = parsers.fail
2351     syntax.HtmlEntity = parsers.fail
2352 end
2353
2354 if not options.inlineFootnotes then
2355     syntax.InlineNote = parsers.fail
2356 end
2357
2358 if not options.smartEllipses then
2359     syntax.Smart = parsers.fail
2360 end
2361
2362 local blocks_toplevel_t = util.table_copy(syntax)
2363 blocks_toplevel_t.Paragraph = larsers.ToplevelParagraph

```

```

2364 larsers.blocks_toplevel = Ct(blocks_toplevel_t)
2365
2366 larsers.blocks = Ct(syntax)
2367
2368 local inlines_t = util.table_copy(syntax)
2369 inlines_t[1] = "Inlines"
2370 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
2371 larsers.inlines = Ct(inlines_t)
2372
2373 local inlines_no_link_t = util.table_copy(inlines_t)
2374 inlines_no_link_t.Link = parsers.fail
2375 larsers.inlines_no_link = Ct(inlines_no_link_t)
2376
2377 local inlines_no_inline_note_t = util.table_copy(inlines_t)
2378 inlines_no_inline_note_t.InlineNote = parsers.fail
2379 larsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
2380
2381 local inlines_nbsp_t = util.table_copy(inlines_t)
2382 inlines_nbsp_t.Endline = larsers.NonbreakingEndline
2383 inlines_nbsp_t.Space = larsers.NonbreakingSpace
2384 larsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

3.1.5.14 Exported Conversion Function Define `reader->convert` as a function that converts markdown string `input` into a plain \TeX output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

2385 function self.convert(input)
2386     references = {}

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.2). The `cacheDir` option is disregarded.

```

2387     local opt_string = {}
2388     for k,_ in pairs(defaultOptions) do
2389         local v = options[k]
2390         if k ~= "cacheDir" then
2391             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
2392         end
2393     end
2394     table.sort(opt_string)
2395     local salt = table.concat(opt_string, ",") .. "," .. metadata.version

```

Produce the cache file, transform its filename via the `writer->pack` method, and return the result.

```

2396     local name = util.cache(options.cacheDir, input, salt, function(input)
2397         return util.ropetostring(parse_blocks_toplevel(input)) .. writer.eof
2398     end, ".md" .. writer.suffix)

```

```

2399     return writer.pack(name)
2400 end
2401 return self
2402 end

```

3.1.6 Conversion from Markdown to Plain T_EX

The `new` method returns the `reader->convert` function of a reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object associated with `options`.

```

2403 function M.new(options)
2404     local writer = M.writer.new(options)
2405     local reader = M.reader.new(writer, options)
2406     return reader.convert
2407 end
2408
2409 return M

```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```

2410 \def\markdownInfo#1{%
2411     \immediate\write-1{(l.\the\inputlineno) markdown.tex info: #1.}}%
2412 \def\markdownWarning#1{%
2413     \immediate\write16{(l.\the\inputlineno) markdown.tex warning: #1}}%
2414 \def\markdownError#1#2{%
2415     \errhelp{#2.}%
2416     \errmessage{(l.\the\inputlineno) markdown.tex error: #1}}%

```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

2417 \def\markdownRendererInterblockSeparatorPrototype{\par}%
2418 \def\markdownRendererLineBreakPrototype{\hfil\break}%
2419 \let\markdownRendererEllipsisPrototype\dots
2420 \def\markdownRendererNbspPrototype{~}%
2421 \def\markdownRendererLeftBracePrototype{\char'{}%
2422 \def\markdownRendererRightBracePrototype{\char'}%
2423 \def\markdownRendererDollarSignPrototype{\char'$}%
2424 \def\markdownRendererPercentSignPrototype{\char'\}%

```

```

2425 \def\markdownRendererAmpersandPrototype{\char'&}%
2426 \def\markdownRendererUnderscorePrototype{\char'_%}
2427 \def\markdownRendererHashPrototype{\char'#}%
2428 \def\markdownRendererCircumflexPrototype{\char'^}%
2429 \def\markdownRendererBackslashPrototype{\char'\}%
2430 \def\markdownRendererTildePrototype{\char'~}%
2431 \def\markdownRendererPipePrototype{|}%
2432 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
2433 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
2434 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
2435   \markdownInput{#3}}%
2436 \def\markdownRendererContentBlockOnlineImagePrototype{%
2437   \markdownRendererImage}%
2438 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
2439   \markdownRendererInputFencedCode{#3}{#2}}%
2440 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
2441 \def\markdownRendererUlBeginPrototype{}%
2442 \def\markdownRendererUlBeginTightPrototype{}%
2443 \def\markdownRendererUlItemPrototype{}%
2444 \def\markdownRendererUlItemEndPrototype{}%
2445 \def\markdownRendererUlEndPrototype{}%
2446 \def\markdownRendererUlEndTightPrototype{}%
2447 \def\markdownRendererOlBeginPrototype{}%
2448 \def\markdownRendererOlBeginTightPrototype{}%
2449 \def\markdownRendererOlItemPrototype{}%
2450 \def\markdownRendererOlItemWithNumberPrototype#1{}%
2451 \def\markdownRendererOlItemEndPrototype{}%
2452 \def\markdownRendererOlEndPrototype{}%
2453 \def\markdownRendererOlEndTightPrototype{}%
2454 \def\markdownRendererDlBeginPrototype{}%
2455 \def\markdownRendererDlBeginTightPrototype{}%
2456 \def\markdownRendererDlItemPrototype#1{#1}%
2457 \def\markdownRendererDlItemEndPrototype{}%
2458 \def\markdownRendererDlDefinitionBeginPrototype{}%
2459 \def\markdownRendererDlDefinitionEndPrototype{\par}%
2460 \def\markdownRendererDlEndPrototype{}%
2461 \def\markdownRendererDlEndTightPrototype{}%
2462 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
2463 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
2464 \def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
2465 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
2466 \def\markdownRendererInputVerbatimPrototype#1{%
2467   \par{\tt\input"#1"\relax}\par}%
2468 \def\markdownRendererInputFencedCodePrototype#1#2{%
2469   \markdownRendererInputVerbatimPrototype{#1}}%
2470 \def\markdownRendererHeadingOnePrototype#1{#1}%
2471 \def\markdownRendererHeadingTwoPrototype#1{#1}%

```

```

2472 \def\markdownRendererHeadingThreePrototype#1{#1}%
2473 \def\markdownRendererHeadingFourPrototype#1{#1}%
2474 \def\markdownRendererHeadingFivePrototype#1{#1}%
2475 \def\markdownRendererHeadingSixPrototype#1{#1}%
2476 \def\markdownRendererHorizontalRulePrototype{}%
2477 \def\markdownRendererFootnotePrototype#1{#1}%
2478 \def\markdownRendererCitePrototype#1{}%
2479 \def\markdownRendererTextCitePrototype#1{}%

```

3.2.3 Lua Snippets

The `\markdownLuaOptions` macro expands to a Lua table that contains the plain \TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.2).

```

2480 \def\markdownLuaOptions{%
2481 \ifx\markdownOptionBlankBeforeBlockquote\undefined\else
2482   blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
2483 \fi
2484 \ifx\markdownOptionBlankBeforeCodeFence\undefined\else
2485   blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
2486 \fi
2487 \ifx\markdownOptionBlankBeforeHeading\undefined\else
2488   blankBeforeHeading = \markdownOptionBlankBeforeHeading,
2489 \fi
2490 \ifx\markdownOptionBreakableBlockquotes\undefined\else
2491   breakableBlockquotes = \markdownOptionBreakableBlockquotes,
2492 \fi
2493   cacheDir = "\markdownOptionCacheDir",
2494 \ifx\markdownOptionCitations\undefined\else
2495   citations = \markdownOptionCitations,
2496 \fi
2497 \ifx\markdownOptionCitationNbsps\undefined\else
2498   citationNbsps = \markdownOptionCitationNbsps,
2499 \fi
2500 \ifx\markdownOptionCodeSpans\undefined\else
2501   codeSpans = \markdownOptionCodeSpans,
2502 \fi
2503 \ifx\markdownOptionContentBlocks\undefined\else
2504   contentBlocks = \markdownOptionContentBlocks,
2505 \fi
2506 \ifx\markdownOptionContentBlocksLanguageMap\undefined\else
2507   contentBlocksLanguageMap =
2508     "\markdownOptionContentBlocksLanguageMap",
2509 \fi
2510 \ifx\markdownOptionDefinitionLists\undefined\else
2511   definitionLists = \markdownOptionDefinitionLists,
2512 \fi

```

```

2513 \ifx\markdownOptionFootnotes\undefined\else
2514   footnotes = \markdownOptionFootnotes,
2515 \fi
2516 \ifx\markdownOptionFencedCode\undefined\else
2517   fencedCode = \markdownOptionFencedCode,
2518 \fi
2519 \ifx\markdownOptionHashEnumerators\undefined\else
2520   hashEnumerators = \markdownOptionHashEnumerators,
2521 \fi
2522 \ifx\markdownOptionHtml\undefined\else
2523   html = \markdownOptionHtml,
2524 \fi
2525 \ifx\markdownOptionHybrid\undefined\else
2526   hybrid = \markdownOptionHybrid,
2527 \fi
2528 \ifx\markdownOptionInlineFootnotes\undefined\else
2529   inlineFootnotes = \markdownOptionInlineFootnotes,
2530 \fi
2531   outputDir = "\markdownOptionOutputDir",
2532 \ifx\markdownOptionPreserveTabs\undefined\else
2533   preserveTabs = \markdownOptionPreserveTabs,
2534 \fi
2535 \ifx\markdownOptionSmartEllipses\undefined\else
2536   smartEllipses = \markdownOptionSmartEllipses,
2537 \fi
2538 \ifx\markdownOptionStartNumber\undefined\else
2539   startNumber = \markdownOptionStartNumber,
2540 \fi
2541 \ifx\markdownOptionTightLists\undefined\else
2542   tightLists = \markdownOptionTightLists,
2543 \fi
2544 \ifx\markdownOptionUnderscores\undefined\else
2545   underscores = \markdownOptionUnderscores,
2546 \fi}
2547 }%

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain \TeX . It exposes the `convert` function for the use by any further Lua code.

```

2548 \def\markdownPrepare{%

```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```

2549 local lfs = require("lfs")
2550 local cacheDir = "\markdownOptionCacheDir"
2551 if lfs.isdir(cacheDir) == true then else
2552   assert(lfs.mkdir(cacheDir))
2553 end

```

Next, load the `markdown` module and create a converter function using the plain \TeX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
2554 local md = require("markdown")
2555 local convert = md.new(\markdownLuaOptions)
2556 }%
```

3.2.4 Buffering Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
2557 \csname newread\endcsname\markdownInputFileStream
2558 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
2559 \begingroup
2560 \catcode'\^^I=12%
2561 \gdef\markdownReadAndConvertTab{^^I}%
2562 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the \TeX_{ϵ} `\filecontents` macro to plain \TeX .

```
2563 \begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition.

```
2564 \catcode'\^^M=13%
2565 \catcode'\^^I=13%
2566 \catcode'|=0%
2567 \catcode'\=12%
2568 |gdef|markdownReadAndConvert#1#2{%
2569 |begingroup%
```

Open the `\markdownOptionInputTempFileName` file for writing.

```
2570 |immediate|openout|markdownOutputFileStream%
2571 |markdownOptionInputTempFileName%
2572 |markdownInfo{Buffering markdown input into the temporary %
2573 input file "|markdownOptionInputTempFileName" and scanning %
2574 for the closing token sequence "#1"}%
```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
2575 |def|do##1{|catcode'##1=12}|dospecials%
2576 |catcode'|=12%
2577 |markdownMakeOther%
```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Note the use of the comments to ensure that the entire macro is at a single line and therefore no (active) newline symbols are produced.

```
2578 |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{%
```

When the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file.

```
2579 |ifx|relax##3|relax%
2580 |immediate|write|markdownOutputFileStream{##1}%
2581 |else%
```

When the ending token sequence appears in the line, make the next newline character close the `\markdownOptionInputTempFileName` file, return the character categories back to the former state, convert the `\markdownOptionInputTempFileName` file from markdown to plain TeX, `\input` the result of the conversion, and expand the ending control sequence.

```
2582 |def^^M{%
2583 |markdownInfo{The ending token sequence was found}%
2584 |immediate|closeout|markdownOutputFileStream%
2585 |endgroup%
2586 |markdownInput|markdownOptionInputTempFileName%
2587 #2}%
2588 |fi%
```

Repeat with the next line.

```
2589 ^^M}%
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
2590 |catcode'|^^I=13%
2591 |def^^I{|markdownReadAndConvertTab}%
```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```
2592 |catcode'|^^M=13%
2593 |def^^M##1^^M{%
2594 |def^^M####1^^M{%
2595 |markdownReadAndConvertProcessLine####1#1#1|relax}%
2596 ^^M}%
2597 ^^M}%
```

Reset the character categories back to the former state.

```
2598 |endgroup
```


3.2.5 Lua Shell Escape Bridge

The following T_EX code is intended for T_EX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of 0 and 1.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the LuaT_EX engine, their TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the T_EXLua interpreter (see [1, Section 3.1.1]).

```
2599 \ifnum\markdownMode<2\relax
2600 \ifnum\markdownMode=0\relax
2601   \markdownInfo{Using mode 0: Shell escape via write18}%
2602 \else
2603   \markdownInfo{Using mode 1: Shell escape via os.execute}%
2604 \fi
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (1), disabled (0), or restricted (2).

Inherit the value of the the `\pdfshellescape` (LuaT_EX, PdfT_EX) or the `\shellescape` (X_YT_EX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```
2605 \ifx\pdfshellescape\undefined
2606   \ifx\shellescape\undefined
2607     \ifnum\markdownMode=0\relax
2608       \def\markdownExecuteShellEscape{1}%
2609     \else
2610       \def\markdownExecuteShellEscape{%
2611         \directlua{tex.sprint(status.shell_escape or "1")}}%
2612     \fi
2613   \else
2614     \let\markdownExecuteShellEscape\shellescape
2615   \fi
2616 \else
2617   \let\markdownExecuteShellEscape\pdfshellescape
2618 \fi
```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```
2619 \ifnum\markdownMode=0\relax
2620   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
2621 \else
2622   \def\markdownExecuteDirect#1{%
2623     \directlua{os.execute("\luaescapestring{#1}")}}%
```

2624 \fi

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```
2625 \def\markdownExecute#1{%
2626   \ifnum\markdownExecuteShellEscape=1\relax
2627     \markdownExecuteDirect{#1}%
2628   \else
2629     \markdownError{I can not access the shell}{Either run the TeX
2630       compiler with the --shell-escape or the --enable-write18 flag,
2631       or set shell_escape=t in the texmf.cnf file}%
2632   \fi}%
```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TeX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

2633 \begingroup

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```
2634 \catcode'\=0%
2635 \catcode'\=12%
2636 \gdef\markdownLuaExecute#1{%
```

Create the file `\markdownOptionHelperScriptFileName` and fill it with the input Lua code prepended with `kpathsea` initialization, so that Lua modules from the TeX distribution are available.

```
2637   |immediate|openout|markdownOutputFileStream=%
2638   |markdownOptionHelperScriptFileName
2639   |markdownInfo{Writing a helper Lua script to the file
2640   "|markdownOptionHelperScriptFileName"}%
2641   |immediate|write|markdownOutputFileStream{%
2642     local ran_ok, error = pcall(function()
2643       local kpse = require('kpse')
2644       kpse.set_program_name('luatex')
2645       #1
2646     end)
```

If there was an error, use the file `\markdownOptionErrorTempFileName` to store the error message.

```
2647     if not ran_ok then
2648       local file = io.open("%
2649       |markdownOptionOutputDir
2650       /|markdownOptionErrorTempFileName", "w")
2651       if file then
2652         file:write(error .. "\n")
2653         file:close()
```

```

2654         end
2655         print('\markdownError{An error was encountered while executing
2656             Lua code}{For further clues, examine the file
2657             "|markdownOptionOutputDir
2658             /|markdownOptionErrorTempFileName"}')
2659     end}%
2660 |immediate|closeout|markdownOutputFileStream

```

Execute the generated `\markdownOptionHelperScriptFileName` Lua script using the \TeX Lua binary and store the output in the `\markdownOptionOutputTempFileName` file.

```

2661 |markdownInfo{Executing a helper Lua script from the file
2662     "|markdownOptionHelperScriptFileName" and storing the result in the
2663     file "|markdownOptionOutputTempFileName"}%
2664 |markdownExecute{texlua "|markdownOptionOutputDir
2665     /|markdownOptionHelperScriptFileName" > %
2666     "|markdownOptionOutputDir
2667     /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `\markdownOptionOutputTempFileName` file.

```

2668 |input|markdownOptionOutputTempFileName|relax}%
2669 |endgroup

```

3.2.6 Direct Lua Access

The following \TeX code is intended for \TeX engines that provide direct access to Lua (Lua \TeX). The macro `\markdownLuaExecute` defined here and in Section 3.2.5 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

2670 \else
2671 \markdownInfo{Using mode 2: Direct Lua access}%

```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `\tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.5,

```

2672 \def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%
2673 \fi

```

3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain \TeX .

```

2674 \begingroup

```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```
2675 \catcode'\=0%
2676 \catcode'\=12%
2677 |gdef|markdownInput#1{%
2678   |markdownInfo{Including markdown document "#1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as \LaTeX Mk to track changes to the markdown document.

```
2679   |openin|markdownInputFileStream#1
2680   |closein|markdownInputFileStream
2681   |markdownLuaExecute{%
2682     |markdownPrepare
2683     local input = assert(io.open("#1","r")):read("*a")
```

Since the Lua converter expects UNIX line endings, normalize the input.

```
2684       print(convert(input:gsub("\r\n?", "\n")))}%
2685 |endgroup
```

3.3 \LaTeX Implementation

The \LaTeX implementation makes use of the fact that, apart from some subtle differences, \LaTeX implements the majority of the plain \TeX format (see [5, Section 9]). As a consequence, we can directly reuse the existing plain \TeX implementation.

```
2686 \input markdown
2687 \def\markdownVersionSpace{ }%
2688 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
2689   \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Logging Facilities

The \LaTeX implementation redefines the plain \TeX logging macros (see Section 3.2.1) to use the \LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```
2690 \renewcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2691 \renewcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2692 \renewcommand\markdownError[2]{\PackageError{markdown}{#1}{#2}}%
```

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain \TeX implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```
2693 \let\markdownInputPlainTeX\markdownInput
```

```

2694 \renewcommand\markdownInput[2][{}]{%
2695   \begingroup
2696     \markdownSetup{#1}%
2697     \markdownInputPlainTeX{#2}%
2698   \endgroup}%

```

The `markdown`, and `markdown*` \TeX environments are implemented using the `\markdownReadAndConvert` macro.

```

2699 \renewenvironment{markdown}{%
2700   \markdownReadAndConvert@markdown{}}\relax
2701 \renewenvironment{markdown*}[1]{%
2702   \markdownSetup{#1}%
2703   \markdownReadAndConvert@markdown*}\relax
2704 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

2705   \catcode'\|=0\catcode'\<=1\catcode'\>=2%
2706   \catcode'\|=12\catcode'\{=12\catcode'\}=12%
2707   \gdef\markdownReadAndConvert@markdown#1<%
2708     \markdownReadAndConvert<\end{markdown#1}>%
2709     <|\end<markdown#1>>>%
2710 \endgroup

```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

2711 \DeclareOption*{%
2712   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
2713 \ProcessOptions\relax

```

After processing the options, activate the `renderers` and `rendererPrototypes` keys.

```

2714 \define@key{markdownOptions}{renderers}{%
2715   \setkeys{markdownRenderers}{#1}%
2716   \def\KV@prefix{KV@markdownOptions@}}%
2717 \define@key{markdownOptions}{rendererPrototypes}{%
2718   \setkeys{markdownRendererPrototypes}{#1}%
2719   \def\KV@prefix{KV@markdownOptions@}}%

```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

If the `\markdownOptionTightLists` macro expands to `false`, do not load the `paralist` package. This is necessary for $\TeX 2_{\epsilon}$ document classes that do not play nice

with paralist, such as beamer. If the `\markdownOptionTightLists` is undefined and the beamer document class is in use, then do not load the paralist package either.

```

2720 \ifx\markdownOptionTightLists\undefined
2721   \@ifclassloaded{beamer}{}{
2722     \RequirePackage{paralist}}
2723 \else
2724   \ifthenelse{\equal{\markdownOptionTightLists}{false}}{}{
2725     \RequirePackage{paralist}}
2726 \fi

```

If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

2727 \@ifpackageloaded{paralist}{
2728   \markdownSetup{rendererPrototypes={
2729     ulBeginTight = {\begin{compactitem}},
2730     ulEndTight   = {\end{compactitem}},
2731     olBeginTight = {\begin{compactenum}},
2732     olEndTight   = {\end{compactenum}},
2733     dlBeginTight = {\begin{compactdesc}},
2734     dlEndTight   = {\end{compactdesc}}}}
2735 }{
2736   \markdownSetup{rendererPrototypes={
2737     ulBeginTight = {\markdownRendererUlBegin},
2738     ulEndTight   = {\markdownRendererUlEnd},
2739     olBeginTight = {\markdownRendererOlBegin},
2740     olEndTight   = {\markdownRendererOlEnd},
2741     dlBeginTight = {\markdownRendererDlBegin},
2742     dlEndTight   = {\markdownRendererDlEnd}}}}
2743 \markdownSetup{rendererPrototypes={
2744   lineBreak = {\},
2745   leftBrace = {\textbraceleft},
2746   rightBrace = {\textbraceright},
2747   dollarSign = {\textdollar},
2748   underscore = {\textunderscore},
2749   circumflex = {\textasciicircum},
2750   backslash = {\textbackslash},
2751   tilde = {\textasciitilde},
2752   pipe = {\textbar},
2753   codeSpan = {\texttt{#1}},
2754   link = {#1\footnote{\ifx\empty#4\empty\else#4:
2755     \fi\texttt{<\url{#3}\texttt{>}}},
2756   contentBlock = {%
2757     \ifthenelse{\equal{#1}{csv}}{%
2758       \begin{table}%
2759       \begin{center}%
2760         \csvautotabular{#3}%

```

```

2761         \end{center}
2762         \ifx\empty#4\empty\else
2763             \caption{#4}%
2764         \fi
2765         \label{tab:#1}%
2766     \end{table}}{%
2767     \markdownInput{#3}}},
2768 image = {%
2769     \begin{figure}%
2770     \begin{center}%
2771         \includegraphics{#3}%
2772     \end{center}%
2773     \ifx\empty#4\empty\else
2774         \caption{#4}%
2775     \fi
2776     \label{fig:#1}%
2777 \end{figure}},
2778 ulBegin = {\begin{itemize}},
2779 ulItem = {\item},
2780 ulEnd = {\end{itemize}},
2781 olBegin = {\begin{enumerate}},
2782 olItem = {\item},
2783 olItemWithNumber = {\item[#1.]},
2784 olEnd = {\end{enumerate}},
2785 dlBegin = {\begin{description}},
2786 dlItem = {\item[#1]},
2787 dlEnd = {\end{description}},
2788 emphasis = {\emph{#1}},
2789 blockQuoteBegin = {\begin{quotation}},
2790 blockQuoteEnd = {\end{quotation}},
2791 inputVerbatim = {\VerbatimInput{#1}},
2792 inputFencedCode = {%
2793     \ifx\relax#2\relax
2794         \VerbatimInput{#1}%
2795     \else
2796         \ifx\minted@jobname\undefined
2797             \ifx\lst@version\undefined
2798                 \markdownRendererInputFencedCode{#1}{}%

```

When the listings package is loaded, use it for syntax highlighting.

```

2799     \else
2800         \lstinputlisting[language=#2]{#1}%
2801     \fi

```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```

2802     \else
2803         \inputminted{#2}{#1}%

```

```

2804     \fi
2805     \fi},
2806     horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
2807     footnote = {\footnote{#1}}}}

```

Support the nesting of strong emphasis.

```

2808 \newif\ifmarkdownLATEXStrongEmphasisNested
2809 \markdownLATEXStrongEmphasisNestedfalse
2810 \markdownSetup{rendererPrototypes={
2811     strongEmphasis = {%
2812         \ifmarkdownLATEXStrongEmphasisNested
2813             \markdownLATEXStrongEmphasisNestedfalse
2814             \textmd{#1}%
2815         \markdownLATEXStrongEmphasisNestedtrue
2816     }else
2817         \markdownLATEXStrongEmphasisNestedtrue
2818         \textbf{#1}%
2819     \markdownLATEXStrongEmphasisNestedfalse
2820 \fi}}

```

Support \LaTeX document classes that do not provide chapters.

```

2821 \ifx\chapter\undefined
2822     \markdownSetup{rendererPrototypes = {
2823         headingOne = {\section{#1}},
2824         headingTwo = {\subsection{#1}},
2825         headingThree = {\subsubsection{#1}},
2826         headingFour = {\paragraph{#1}},
2827         headingFive = {\subparagraph{#1}}}}
2828 \else
2829     \markdownSetup{rendererPrototypes = {
2830         headingOne = {\chapter{#1}},
2831         headingTwo = {\section{#1}},
2832         headingThree = {\subsection{#1}},
2833         headingFour = {\subsubsection{#1}},
2834         headingFive = {\paragraph{#1}},
2835         headingSix = {\subparagraph{#1}}}}
2836 \fi

```

There is a basic implementation for citations that uses the \LaTeX `\cite` macro. There is also a more advanced implementation that uses the Bib \LaTeX `\autocites` and `\textcites` macros. This implementation will be used, when Bib \LaTeX is loaded.

```

2837 \newcount\markdownLaTeXCitationsCounter
2838
2839 % Basic implementation
2840 \def\markdownLaTeXBasicCitations#1#2#3#4{%
2841     \advance\markdownLaTeXCitationsCounter by 1\relax
2842     \ifx\relax#2\relax\else#2~\fi\cite[#3]{#4}%
2843     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```



```

2844 \expandafter\@gobble
2845 \fi\markdownLaTeXBasicCitations}
2846 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
2847
2848 % BibLaTeX implementation
2849 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
2850 \advance\markdownLaTeXCitationsCounter by 1\relax
2851 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2852 \autocites#1[#3][#4]{#5}%
2853 \expandafter\@gobbletwo
2854 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
2855 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
2856 \advance\markdownLaTeXCitationsCounter by 1\relax
2857 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2858 \textcites#1[#3][#4]{#5}%
2859 \expandafter\@gobbletwo
2860 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
2861
2862 \markdownSetup{rendererPrototypes = {
2863 cite = {%
2864 \markdownLaTeXCitationsCounter=1%
2865 \def\markdownLaTeXCitationsTotal{#1}%
2866 \ifx\autocites\undefined
2867 \expandafter
2868 \markdownLaTeXBasicCitations
2869 \else
2870 \expandafter\expandafter\expandafter
2871 \markdownLaTeXBibLaTeXCitations
2872 \expandafter{\expandafter}%
2873 \fi},
2874 textCite = {%
2875 \markdownLaTeXCitationsCounter=1%
2876 \def\markdownLaTeXCitationsTotal{#1}%
2877 \ifx\textcites\undefined
2878 \expandafter
2879 \markdownLaTeXBasicTextCitations
2880 \else
2881 \expandafter\expandafter\expandafter
2882 \markdownLaTeXBibLaTeXTextCitations
2883 \expandafter{\expandafter}%
2884 \fi}}}

```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the

`\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```
2885 \newcommand\markdownMakeOther{%
2886   \count0=128\relax
2887   \loop
2888     \catcode\count0=11\relax
2889     \advance\count0 by 1\relax
2890   \ifnum\count0<256\repeat}%
```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain T_EX implementation. As a consequence, we can directly reuse the existing plain T_EX implementation after supplying the missing plain T_EX macros.

```
2891 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2892   \do\#\do\~\do\_ \do\% \do\~}%
2893 \input markdown
```

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents LaTeX package.

```
2894 \def\markdownMakeOther{%
2895   \count0=128\relax
2896   \loop
2897     \catcode\count0=11\relax
2898     \advance\count0 by 1\relax
2899   \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConT_EXt.

```
2900   \catcode'|=12}%
```

3.4.1 Logging Facilities

The ConT_EXt implementation redefines the plain T_EX logging macros (see Section 3.2.1) to use the ConT_EXt `\writestatus` macro.

```
2901 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2902 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
```

3.4.2 Typesetting Markdown

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

```
2903 \begingroup
```

Locally swap the category code of the backslash symbol with the pipe symbol. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```
2904 \catcode'\|=0%
2905 \catcode'\|=12%
2906 |gdef|startmarkdown{%
2907     |markdownReadAndConvert{\stopmarkdown}%
2908                               {|stopmarkdown}}%
2909 |endgroup
```

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
2910 \def\markdownRendererLineBreakPrototype{\blank}%
2911 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
2912 \def\markdownRendererRightBracePrototype{\textbraceright}%
2913 \def\markdownRendererDollarSignPrototype{\textdollar}%
2914 \def\markdownRendererPercentSignPrototype{\percent}%
2915 \def\markdownRendererUnderscorePrototype{\textunderscore}%
2916 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
2917 \def\markdownRendererBackslashPrototype{\textbackslash}%
2918 \def\markdownRendererTildePrototype{\textasciitilde}%
2919 \def\markdownRendererPipePrototype{\char' |}%
2920 \def\markdownRendererLinkPrototype#1#2#3#4{%
2921     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
2922     \fi\texttt<\hyphenatedurl{#3}>}}%
2923 \usemodule[database]
2924 \defineseparatedlist
2925     [MarkdownConTeXtCSV]
2926     [separator={,},
2927     before=\bTABLE,after=\eTABLE,
2928     first=\bTR,last=\eTR,
2929     left=\bTD,right=\eTD]
2930 \def\markdownConTeXtCSV{csv}
2931 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
2932     \def\markdownConTeXtCSV@arg{#1}%
2933     \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
2934         \placetable[] [tab:#1]{#4}{%
2935             \processseparatedfile[MarkdownConTeXtCSV][#3]}%
2936     \else
```

```

2937 \markdownInput{#3}%
2938 \fi}%
2939 \def\markdownRendererImagePrototype#1#2#3#4{%
2940   \placefigure[] [fig:#1]{#4}{\externalfigure[#3]}}%
2941 \def\markdownRendererUlBeginPrototype{\startitemize}%
2942 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
2943 \def\markdownRendererUlItemPrototype{\item}%
2944 \def\markdownRendererUlEndPrototype{\stopitemize}%
2945 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
2946 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
2947 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
2948 \def\markdownRendererOlItemPrototype{\item}%
2949 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
2950 \def\markdownRendererOlEndPrototype{\stopitemize}%
2951 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
2952 \definedescription
2953   [MarkdownConTeXtDlItemPrototype]
2954   [location=hanging,
2955    margin=standard,
2956    headstyle=bold]%
2957 \definestartstop
2958   [MarkdownConTeXtDlPrototype]
2959   [before=\blank,
2960    after=\blank]%
2961 \definestartstop
2962   [MarkdownConTeXtDlTightPrototype]
2963   [before=\blank\startpacked,
2964    after=\stoppacked\blank]%
2965 \def\markdownRendererDlBeginPrototype{%
2966   \startMarkdownConTeXtDlPrototype}%
2967 \def\markdownRendererDlBeginTightPrototype{%
2968   \startMarkdownConTeXtDlTightPrototype}%
2969 \def\markdownRendererDlItemPrototype#1{%
2970   \startMarkdownConTeXtDlItemPrototype{#1}}%
2971 \def\markdownRendererDlItemEndPrototype{%
2972   \stopMarkdownConTeXtDlItemPrototype}%
2973 \def\markdownRendererDlEndPrototype{%
2974   \stopMarkdownConTeXtDlPrototype}%
2975 \def\markdownRendererDlEndTightPrototype{%
2976   \stopMarkdownConTeXtDlTightPrototype}%
2977 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
2978 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
2979 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
2980 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
2981 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
2982 \def\markdownRendererInputFencedCodePrototype#1#2{%
2983   \ifx\relax#2\relax

```

```

2984     \typefile{#1}%
2985     \else

```

The code fence infostring is used as a name from the ConT_EXt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```

% Map the 'TEX' syntax highlighter to the 'latex' infostring.
\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
  ~~~ latex
  \documentclass{article}
  \begin{document}
    Hello world!
  \end{document}
  ~~~
  \stopmarkdown
\stoptext

```

```

2986     \typefile{#2} []{#1}%
2987     \fi}%
2988 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
2989 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
2990 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
2991 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
2992 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
2993 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
2994 \def\markdownRendererHorizontalRulePrototype{%
2995   \blackrule[height=1pt, width=\hsize]}%
2996 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
2997 \stopmodule\protect

```

References

1. LUAT_EX DEVELOPMENT TEAM. *LuaT_EX reference manual* [online]. 2017 [visited on 2018-01-08]. Available from: <http://www.luatex.org/svn/trunk/manual/luatex.pdf>.
2. SOTKOV, Anton. *File transclusion syntax for Markdown* [online]. 2017 [visited on 2018-01-08]. Available from: <https://github.com/iainc/Markdown-Content-Blocks>.
3. KNUTH, Donald Ervin. *The T_EXbook*. 3rd ed. Addison-Wesley, 1986. ISBN 0-201-13447-0.

4. IERUSALIMSKY, Roberto. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. ISBN 978-85-903798-5-0.
5. BRAAMS, Johannes; CARLISLE, David; JEFFREY, Alan; LAMPORT, Leslie; MITTELBAACH, Frank; ROWLEY, Chris; SCHÖPF, Rainer. *The \LaTeX 2_ε Sources* [online]. 2017 [visited on 2018-01-08]. Available from: <http://mirrors.ctan.org/macros/latex/base/source2e.pdf>.