

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.17.1-0-g8ca83f5
2022-10-03

Contents

1	Introduction	1	3	Implementation	100
1.1	Requirements	2	3.1	Lua Implementation . . .	100
1.2	Feedback	5	3.2	Plain T _E X Implementation	221
1.3	Acknowledgements	5	3.3	L ^A T _E X Implementation . .	238
2	Interfaces	6	3.4	ConT _E Xt Implementation	261
2.1	Lua Interface	6			
2.2	Plain T _E X Interface	39			
2.3	L ^A T _E X Interface	81			
2.4	ConT _E Xt Interface	97			
				References	266
				Index	267

List of Figures

1	A block diagram of the Markdown package	7
2	A sequence diagram of typesetting a document using the T _E X interface . .	36
3	A sequence diagram of typesetting a document using the Lua CLI	36
4	Various formats of mathematical formulae	88
5	The banner of the Markdown package	89
6	A pushdown automaton that recognizes T _E X comments	163

1 Introduction

The Markdown package¹ converts markdown² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the

¹See <https://ctan.org/pkg/markdown>.

²See <https://daringfireball.net/projects/markdown/basics>.

implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.³

```
1 local metadata = {
2   version   = "(((VERSION)))",
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               "2016-2022 Vít Novotný"},
7   license   = "LPPL 1.3c"
8 }
9
10 if not modules then modules = { } end
11 modules['markdown'] = metadata
```

1.1 Requirements

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

1.1.1 Lua Requirements

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

LPeg ≥ 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 ≥ 0.10 is included in LuaTeX $\geq 0.72.0$ (TeXLive ≥ 2013).

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

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 LuaTeX (TeXLive ≥ 2008).

```
13 local unicode
14 (function()
15   local ran_ok
16   ran_ok, unicode = pcall(require, "unicode")
```

If the Selene Unicode library is unavailable and we are using Lua ≥ 5.3 , we will use the built-in support for Unicode.

³See <http://mirrors.ctan.org/macros/generic/markdown/markdown.html>.

```

17   if not ran_ok then
18       unicode = {"utf8"}={char=utf8.char}}
19   end
20 end()

```

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 LuaTeX (TeXLive \geq 2008).

```

21 local md5 = require("md5")

```

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [1, Section 4.3]. Beside these, we also carry the following third-party Lua libraries:

api7/lua-tinyyaml A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jekyllData` option is enabled.

1.1.2 Plain TeX Requirements

The plain TeX part of the package requires that the plain TeX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.1.1), and the following packages:

expl3 A package that enables the expl3 language from the L^AT_EX3 kernel in TeX Live \leq 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```

22 <@@=markdown>
23 \ifx\ExplSyntaxOn\undefined
24   \input expl3-generic\relax
25 \fi

```

lt3luabridge A package that allows us to execute Lua code with LuaTeX as well as with other TeX engines that provide the *shell escape* capability, which allows them to execute code with the system's shell.

The plain TeX part of the package also requires 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 TeX 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 LuaTeX (TeXLive \geq 2008). The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [1, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 4.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.6), the plain TeX part of the package will require that either the LuaTeX `\directlua` primitive or the shell access file stream 18 is available in your TeX engine. If only the shell access file stream is available in your TeX engine (as is the case with pdfTeX and XeTeX) or if you enforce the use of shell using the `\markdownMode` macro, then unless your TeX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 L^ATeX Requirements

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

```
26 \NeedsTeXFormat{LaTeX2e}%
```

a TeX engine that extends ε -TeX, and all the plain TeX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.4 and 3.3.4) or L^ATeX themes (see Section 2.3.2.2) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.1):

url A package that provides the `\url` macro for the typesetting of links.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists.

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L^ATeX theme (see Section 2.3.2.2), and to provide default token renderer prototypes.

fancyvrb A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code.

csvsimple A package that provides the `\csvautotabular` macro for typesetting csv files in the default renderer prototypes for iA,Writer content blocks.

gobble A package that provides the `\@gobblethree` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive \geq 2016.

amsmath and amssymb Packages that provide symbols used for drawing ticked and unticked boxes.

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L^AT_EX theme, see Section 2.3.2.2.

grffile A package that extends the name processing of package graphics to support a larger range of file names in $2006 \leq \text{T_EX Live} \leq 2019$. Since $\text{T_EX Live} \geq 2020$, the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes, see Section 2.3.2.2.

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.6, and also in the default renderer prototype for attribute identifiers.

soulutf8 A package that is used in the default renderer prototype for strike-throughs.

```
27 \RequirePackage{expl3}
```

1.1.4 ConT_EXt Prerequisites

The ConT_EXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T_EX prerequisites (see Section 1.1.2), and the following ConT_EXt 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).

1.2 Feedback

Please use the Markdown project page on GitHub⁴ to report bugs and submit feature requests. 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 to the T_EX-L^AT_EX Stack Exchange.⁵ community question answering web site under the [markdown](#) tag.

1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

⁴See <https://github.com/witiko/markdown/issues>.

⁵See <https://tex.stackexchange.com>.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

Support for content slicing (Lua options `shiftHeadings` and `slice`) and pipe tables (Lua options `pipeTables` and `tableCaptions`) was graciously sponsored by David Vins and Omedym.

The $\text{T}_{\text{E}}\text{X}$ implementation of the package draws inspiration from several sources including the source code of $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from $\text{T}_{\text{E}}\text{X}$, the filecontents package by Scott Pakin and others.

2 Interfaces

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

Since neither $\text{T}_{\text{E}}\text{X}$ nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to $\text{T}_{\text{E}}\text{X}$ *token renderers* is exposed by the Lua layer. The plain $\text{T}_{\text{E}}\text{X}$ layer exposes the conversion capabilities of Lua as $\text{T}_{\text{E}}\text{X}$ macros. The $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ and $\text{ConT}_{\text{E}}\text{Xt}$ layers provide syntactic sugar on top of plain $\text{T}_{\text{E}}\text{X}$ macros. The user can interface with any and all layers.

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain $\text{T}_{\text{E}}\text{X}$. This interface is used by the plain $\text{T}_{\text{E}}\text{X}$ 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.

```
28 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain $\text{T}_{\text{E}}\text{X}$

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain $\text{T}_{\text{E}}\text{X}$ according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

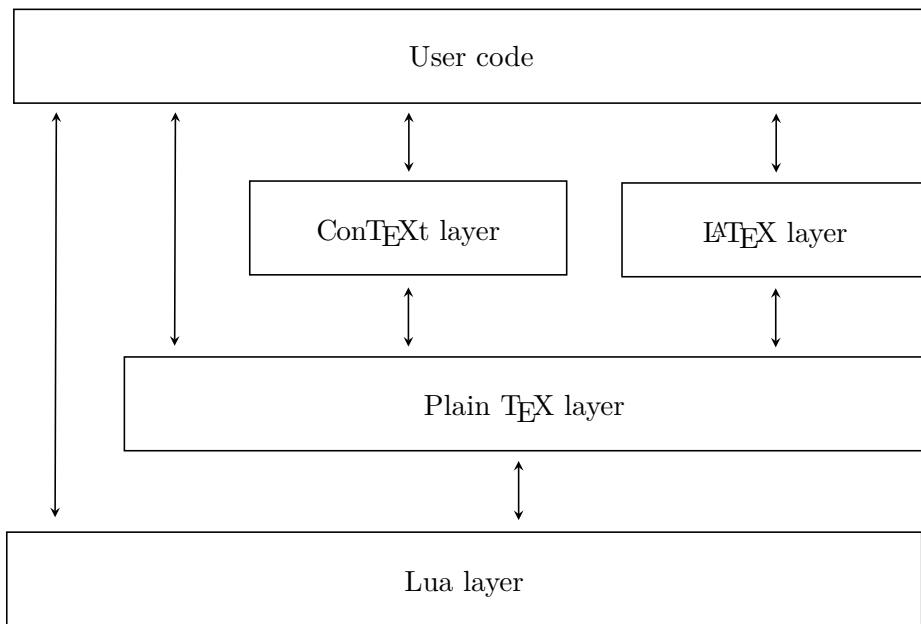


Figure 1: A block diagram of the Markdown package

The following example Lua code converts the markdown string `Hello *world*!` to a T_EX output using the default options and prints the T_EX output:

```

local md = require("markdown")
local convert = md.new()
print(convert("Hello *world*!"))

```

2.1.2 User-Defined Syntax Extensions

For the purpose of user-defined syntax extensions, the Lua interface also exposes the `reader` object, which performs the lexical and syntactic analysis of markdown text and which exposes the `reader->insert_pattern` and `reader->add_special_character` methods for extending the PEG grammar of markdown.

The read-only `walkable_syntax` hash table stores those rules of the PEG grammar of markdown that can be represented as an ordered choice of terminal symbols. These rules can be modified by user-defined syntax extensions.

```

29 local walkable_syntax = {
30   Block = {
31     "Blockquote",
32     "Verbatim",

```

```

33     "HorizontalRule",
34     "BulletList",
35     "OrderedList",
36     "Heading",
37     "DisplayHtml",
38     "Paragraph",
39     "Plain",
40 },
41 Inline = {
42     "Str",
43     "Space",
44     "Endline",
45     "UlOrStarLine",
46     "Strong",
47     "Emph",
48     "Link",
49     "Image",
50     "Code",
51     "AutoLinkUrl",
52     "AutoLinkEmail",
53     "AutoLinkRelativeReference",
54     "InlineHtml",
55     "HtmlEntity",
56     "EscapedChar",
57     "Smart",
58     "Symbol",
59 },
60 }

```

The `reader->insert_pattern` method inserts a PEG pattern into the grammar of markdown. The method receives two mandatory arguments: a selector string in the form "*<left-hand side terminal symbol> <before, after, or instead of> <right-hand side terminal symbol>*" and a PEG pattern to insert, and an optional third argument with a name of the PEG pattern for debugging purposes (see the `debugExtensions` option). The name does not need to be unique and shall not be interpreted by the Markdown package; you can treat it as a comment.

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern` with "`Inline after Emph`" (or "`Inline before Link`") and `pattern` as the arguments.

The `reader->add_special_character` method adds a new character with special meaning to the grammar of markdown. The method receives the character as its only argument.

2.1.3 Options

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

```
61 local defaultOptions = {}
```

To enable the enumeration of Lua options, we will maintain the `\g_@@_lua_options_seq` sequence.

```
62 \ExplSyntaxOn
```

```
63 \seq_new:N \g_@@_lua_options_seq
```

To enable the reflection of default Lua options and their types, we will maintain the `\g_@@_default_lua_options_prop` and `\g_@@_lua_option_types_prop` property lists, respectively.

```
64 \prop_new:N \g_@@_lua_option_types_prop
```

```
65 \prop_new:N \g_@@_default_lua_options_prop
```

```
66 \seq_new:N \g_@@_option_layers_seq
```

```
67 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
```

```
68 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
```

```
69 \cs_new:Nn
```

```
70   \@@_add_lua_option:nnn
```

```
71   {
```

```
72     \@@_add_option:Vnnn
```

```
73     \c_@@_option_layer_lua_tl
```

```
74     { #1 }
```

```
75     { #2 }
```

```
76     { #3 }
```

```
77   }
```

```
78 \cs_new:Nn
```

```
79   \@@_add_option:nnnn
```

```
80   {
```

```
81     \seq_put_right:cn
```

```
82     { g_@@_ #1 _options_seq }
```

```
83     { #2 }
```

```
84     \prop_put:cnn
```

```
85     { g_@@_ #1 _option_types_prop }
```

```
86     { #2 }
```

```
87     { #3 }
```

```
88     \prop_put:cnn
```

```
89     { g_@@_default_ #1 _options_prop }
```

```
90     { #2 }
```

```
91     { #4 }
```

```
92     \@@_typecheck_option:n
```

```
93     { #2 }
```

```
94   }
```

```
95 \cs_generate_variant:Nn
```

```
96   \@@_add_option:nnnn
```

```

97   { Vnnn }
98   \tl_const:Nn \c_@@_option_value_true_tl { true }
99   \tl_const:Nn \c_@@_option_value_false_tl { false }
100  \cs_new:Nn \@@_typecheck_option:n
101    {
102      \@@_get_option_type:nN
103        { #1 }
104        \l_tmpa_tl
105      \str_case_e:Vn
106        \l_tmpa_tl
107        {
108          { \c_@@_option_type_boolean_tl }
109          {
110            \@@_get_option_value:nN
111              { #1 }
112              \l_tmpa_tl
113            \bool_if:nF
114              {
115                \str_if_eq_p:VV
116                  \l_tmpa_tl
117                  \c_@@_option_value_true_tl ||
118                \str_if_eq_p:VV
119                  \l_tmpa_tl
120                  \c_@@_option_value_false_tl
121              }
122              {
123                \msg_error:nnnV
124                  { @@ }
125                  { failed-typecheck-for-boolean-option }
126                  { #1 }
127                  \l_tmpa_tl
128              }
129          }
130      }
131  }
132  \msg_new:nnn
133    { @@ }
134    { failed-typecheck-for-boolean-option }
135    {
136      Option~#1~has~value~#2,~
137      but~a~boolean~(true~or~false)~was~expected.
138    }
139  \cs_generate_variant:Nn
140    \str_case_e:nn
141    { Vn }
142  \cs_generate_variant:Nn
143    \msg_error:nnnn

```

```

144   { nnnV }
145 \seq_new:N \g_@@_option_types_seq
146 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
147 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
148 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
149 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
150 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
151 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
152 \tl_const:Nn \c_@@_option_type_number_tl { number }
153 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
154 \tl_const:Nn \c_@@_option_type_path_tl { path }
155 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
156 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
157 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
158 \tl_const:Nn \c_@@_option_type_string_tl { string }
159 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
160 \cs_new:Nn
161   \@@_get_option_type:nN
162   {
163     \bool_set_false:N
164       \l_tmpa_bool
165     \seq_map_inline:Nn
166       \g_@@_option_layers_seq
167       {
168         \prop_get:cnNT
169           { g_@@_ #1 _option_types_prop }
170           { #1 }
171         \l_tmpa_tl
172         {
173           \bool_set_true:N
174             \l_tmpa_bool
175           \seq_map_break:
176         }
177       }
178     \bool_if:nF
179       \l_tmpa_bool
180     {
181       \msg_error:nnn
182         { @@ }
183         { undefined-option }
184         { #1 }
185     }
186     \seq_if_in:NVF
187       \g_@@_option_types_seq
188       \l_tmpa_tl
189     {
190       \msg_error:nnnV

```

```

191         { @@ }
192         { unknown-option-type }
193         { #1 }
194         \l_tmpa_tl
195     }
196     \tl_set_eq:NN
197     #2
198     \l_tmpa_tl
199 }
200 \msg_new:nnn
201 { @@ }
202 { unknown-option-type }
203 {
204     Option~#1~has~unknown~type~#2.
205 }
206 \msg_new:nnn
207 { @@ }
208 { undefined-option }
209 {
210     Option~#1~is~undefined.
211 }
212 \cs_new:Nn
213 \@@_get_default_option_value:nN
214 {
215     \bool_set_false:N
216     \l_tmpa_bool
217     \seq_map_inline:Nn
218     \g_@@_option_layers_seq
219     {
220         \prop_get:cnNT
221         { g_@@_default_ ##1 _options_prop }
222         { #1 }
223         #2
224         {
225             \bool_set_true:N
226             \l_tmpa_bool
227             \seq_map_break:
228         }
229     }
230     \bool_if:nF
231     \l_tmpa_bool
232     {
233         \msg_error:nnn
234         { @@ }
235         { undefined-option }
236         { #1 }
237     }

```

```

238     }
239 \cs_new:Nn
240   \@@_get_option_value:nN
241   {
242     \@@_option_tl_to_csname:nN
243     { #1 }
244     \l_tmpa_tl
245     \cs_if_free:cTF
246     { \l_tmpa_tl }
247     {
248       \@@_get_default_option_value:nN
249       { #1 }
250       #2
251     }
252   }
253   \@@_get_option_type:nN
254   { #1 }
255   \l_tmpa_tl
256   \str_if_eq:NNTF
257   \c_@@_option_type_counter_tl
258   \l_tmpa_tl
259   {
260     \@@_option_tl_to_csname:nN
261     { #1 }
262     \l_tmpa_tl
263     \tl_set:Nx
264     #2
265     { \the \cs:w \l_tmpa_tl \cs_end: }
266   }
267   {
268     \@@_option_tl_to_csname:nN
269     { #1 }
270     \l_tmpa_tl
271     \tl_set:Nv
272     #2
273     { \l_tmpa_tl }
274   }
275 }
276 }
277 \cs_new:Nn \@@_option_tl_to_csname:nN
278 {
279   \tl_set:Nn
280   \l_tmpa_tl
281   { \str_uppercase:n { #1 } }
282   \tl_set:Nx
283   #2
284   {

```

```

285         markdownOption
286         \tl_head:f { \l_tmpa_tl }
287         \tl_tail:n { #1 }
288     }
289 }

```

2.1.4 File and Directory Names

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

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T_EX 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.

```

290 \@@_add_lua_option:nnn
291   { cacheDir }
292   { path }
293   { \markdownOptionOutputDir / _markdown_\jobname }
294 defaultOptions.cacheDir = "."

```

`contentBlocksLanguageMap`= $\langle filename \rangle$
 default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA,Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.3.11 for more information.

```

295 \@@_add_lua_option:nnn
296   { contentBlocksLanguageMap }
297   { path }
298   { markdown-languages.json }
299 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"

```

`debugExtensionsFileName`= $\langle filename \rangle$ default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section ??) and user-defined syntax extensions (see Section ??) have been applied.

```
300 \@@_add_lua_option:nnn
301   { debugExtensionsFileName }
302   { path }
303   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
304 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

`frozenCacheFileName`= $\langle path \rangle$ default: `frozenCache.tex`

A path to an output file (frozen cache) that will be created when the `finalizeCache` option is enabled and will contain a mapping between an enumeration of markdown documents and their auxiliary cache files.

The frozen cache makes it possible to later typeset a plain \TeX document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain \TeX option. As a result, the plain \TeX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
305 \@@_add_lua_option:nnn
306   { frozenCacheFileName }
307   { path }
308   { \markdownOptionCacheDir / frozenCache.tex }
309 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.5 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.

```
310 \@@_add_lua_option:nnn
311   { blankBeforeBlockquote }
312   { boolean }
313   { false }
314 defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence=true, false` default: false

`true` Require a blank line between a paragraph and the following fenced code block.

`false` Do not require a blank line between a paragraph and the following fenced code block.

```
315 \@@_add_lua_option:nnn
316 { blankBeforeCodeFence }
317 { boolean }
318 { false }

319 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.

```
320 \@@_add_lua_option:nnn
321 { blankBeforeHeading }
322 { boolean }
323 { false }

324 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.

```
325 \@@_add_lua_option:nnn
326 { breakableBlockquotes }
327 { boolean }
328 { false }

329 defaultOptions.breakableBlockquotes = false
```


`citationNbsps=true, false`

default: false

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

```
330 \@@_add_lua_option:nnn
331 { citationNbsps }
332 { boolean }
333 { true }
334 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.

```
335 \@@_add_lua_option:nnn
336 { citations }
337 { boolean }
338 { false }
339 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.``
```

```
340 \@@_add_lua_option:nnn  
341 { codeSpans }  
342 { boolean }  
343 { true }  
  
344 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

true Enable the iA,Writer content blocks syntax extension [3]:

```
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.

```
345 \@@_add_lua_option:nnn  
346 { contentBlocks }  
347 { boolean }  
348 { false }  
  
349 defaultOptions.contentBlocks = false
```

`debugExtensions=true, false`

default: `false`

`true`

: Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section ??) and user-defined syntax extensions (see Section ??) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

`false` Do not produce a JSON file with the PEG grammar of markdown.

```
350 \@@_add_lua_option:nnn
351   { debugExtensions }
352   { boolean }
353   { false }
354 defaultOptions.debugExtensions = false
```

`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.

```
355 \@@_add_lua_option:nnn
356   { definitionLists }
357   { boolean }
358   { false }
359 defaultOptions.definitionLists = false
```

`eagerCache=true, false`

default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

This behavior will always be used if the `finalizeCache` option is enabled.

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

This behavior will only be used when the `finalizeCache` option is disabled. Recursive nesting of markdown document fragments is undefined behavior when `eagerCache` is disabled.

```
360 \@@_add_lua_option:nnn
361   { eagerCache }
362   { boolean }
363   { true }

364 defaultOptions.eagerCache = true
```

`extensions=<filenames>`

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the `kpathsea` library is available, files will be searched for not only in the current working directory but also in the \TeX directory structure.

A user-defined syntax extension is a Lua file in the following format:

```
local strike_through = {
  api_version = 2,
  grammar_version = 1,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
    local function between(p, starter, ender)
      ender = lpeg.B(nonspacechar) * ender
      return (starter * #nonspacechar
        * lpeg.Ct(p * (p - ender)^0) * ender)
    end
```

```

    local read_strike_through = between(
        lpeg.V("Inline"), doubleslashes, doubleslashes
    ) / function(s) return {"\\st{", s, "}"} end

    reader.insert_pattern("Inline after Emph", read_strike_through,
        "StrikeThrough")
    reader.add_special_character("/")
end
}

return strike_through

```

The `api_version` and `grammar_version` fields specify the version of the user-defined syntax extension API and the markdown grammar for which the extension was written. See the current API and grammar versions below:

```

365 metadata.user_extension_api_version = 2
366 metadata.grammar_version = 1

```

Any changes to the syntax extension API or grammar will cause the corresponding current version to be incremented. After Markdown 3.0.0, any changes to the API and the grammar will be either backwards-compatible or constitute a breaking change that will cause the major version of the Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `\luamref{reader}` object, such as the `\luamref{reader->insert_pattern}` and `\luamref{reader->add_special_character}` methods, see Section `<#luauserextensions>`.

```

367 \cs_generate_variant:Nn
368   \@@_add_lua_option:nnn
369   { nnV }
370 \@@_add_lua_option:nnV
371   { extensions }
372   { clist }
373   \c_empty_clist
374 defaultOptions.extensions = {}

```

`expectJekyllData=true, false`

default: `false`

`false` When the `jekyllData` option is enabled, then a markdown document may begin with YAML metadata if and only if the metadata begin with the end-of-directives marker (`---`) and they end with either the end-of-directives or the end-of-document marker (`...`):

```
\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}
```

`true` When the `jekyllData` option is enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```
\documentclass{article}
\usepackage[jekyllData, expectJekyllData]{markdown}
\begin{document}
\begin{markdown}
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
```

```

\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}

```

```

375 \@@_add_lua_option:nnn
376   { expectJekyllData }
377   { boolean }
378   { false }

379 defaultOptions.expectJekyllData = false

```

`fancyLists=true, false`

default: false

true Enable the Pandoc fancy list extension:

```

a) first item
b) second item
c) third item

```

false Disable the Pandoc fancy list extension.

```

380 \@@_add_lua_option:nnn
381   { fancyLists }
382   { boolean }
383   { false }

384 defaultOptions.fancyLists = 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.

```

385 \@@_add_lua_option:nnn
386 { fencedCode }
387 { boolean }
388 { false }
389 defaultOptions.fencedCode = false

```

**finalizeCache=true, false**      default: false

Whether an output file specified with the **frozenCacheFileName** option (frozen cache) that contains a mapping between an enumeration of markdown documents and their auxiliary cache files will be created.

The frozen cache makes it possible to later typeset a plain T<sub>E</sub>X document that contains markdown documents without invoking Lua using the **\markdownOptionFrozenCache** plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

390 \@@_add_lua_option:nnn
391 { finalizeCache }
392 { boolean }
393 { false }
394 defaultOptions.finalizeCache = 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.

```
395 \@@_add_lua_option:nnn
396 { footnotes }
397 { boolean }
398 { false }
399 defaultOptions.footnotes = false
```

**frozenCacheCounter**= $\langle number \rangle$  default: 0

The number of the current markdown document that will be stored in an output file (frozen cache) when the **finalizeCache** is enabled. When the document number is 0, then a new frozen cache will be created. Otherwise, the frozen cache will be appended.

Each frozen cache entry will define a T<sub>E</sub>X macro **\markdownFrozenCache** $\langle number \rangle$  that will typeset markdown document number  $\langle number \rangle$ .

```
400 \@@_add_lua_option:nnn
401 { frozenCacheCounter }
402 { counter }
403 { 0 }
404 defaultOptions.frozenCacheCounter = 0
```

**hardLineBreaks**=true, false default: false

**true**      Interpret all newlines within a paragraph as hard line breaks instead of spaces.

**false**     Interpret all newlines within a paragraph as spaces.

```

405 \@@_add_lua_option:nnn
406 { hardLineBreaks }
407 { boolean }
408 { false }
409 defaultOptions.hardLineBreaks = 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.

```

410 \@@_add_lua_option:nnn
411 { hashEnumerators }
412 { boolean }
413 { false }
414 defaultOptions.hashEnumerators = false

```

`headerAttributes=true, false`

default: false

**true** Enable the assignment of HTML attributes to headings:

```

My first heading {#foo}

My second heading ## {#bar .baz}

Yet another heading {key=value}
=====

```

These HTML attributes have currently no effect other than enabling content slicing, see the [slice](#) option.

**false** Disable the assignment of HTML attributes to headings.

```

415 \@@_add_lua_option:nnn
416 { headerAttributes }
417 { boolean }
418 { false }
419 defaultOptions.headerAttributes = false

```

`html=true, false` default: false

- true** Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions 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.

```
420 \@@_add_lua_option:nnn
421 { html }
422 { boolean }
423 { false }
424 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.

```
425 \@@_add_lua_option:nnn
426 { hybrid }
427 { boolean }
428 { false }
429 defaultOptions.hybrid = false
```

`inlineFootnotes=true, false` default: false

- true** Enable the Pandoc inline footnote syntax extension:

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

`false`      Disable the Pandoc inline footnote syntax extension.

```
430 \@@_add_lua_option:nnn
431 { inlineFootnotes }
432 { boolean }
433 { false }
434 defaultOptions.inlineFootnotes = false
```

`jeekyllData=true, false`      default: `false`

`true`      Enable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML:

```

title: 'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
 This is the abstract.

 It consists of two paragraphs.

```

`false`      Disable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML.

```
435 \@@_add_lua_option:nnn
436 { jeekyllData }
437 { boolean }
438 { false }
439 defaultOptions.jekyllData = false
```

`pipeTables=true, false`      default: `false`

`true`      Enable the PHP Markdown pipe table syntax extension:

Right	Left	Default	Center
-----:	:-----	-----	:-----:
12	12	12	12
123	123	123	123
1	1	1	1

`false`      Disable the PHP Markdown pipe table syntax extension.

```
440 \@@_add_lua_option:nnn
441 { pipeTables }
442 { boolean }
443 { false }
444 defaultOptions.pipeTables = false
```

`preserveTabs=true, false`      default: `false`

`true`      Preserve tabs in code block and fenced code blocks.

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

```
445 \@@_add_lua_option:nnn
446 { preserveTabs }
447 { boolean }
448 { false }
449 defaultOptions.preserveTabs = false
```

`relativeReferences=true, false`      default: `false`

`true`      Enable relative references<sup>6</sup> in autolinks:

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

In this paper, we have discovered that most grandmas would rather eat dinner with their grandchildren than get eaten. Begone, wolf!

`false`      Disable relative references in autolinks.

```
450 \@@_add_lua_option:nnn
451 { relativeReferences }
452 { boolean }
453 { false }
454 defaultOptions.relativeReferences = false
```

---

<sup>6</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`shiftHeadings`= $\langle shift\ amount \rangle$  default: 0

All headings will be shifted by  $\langle shift\ amount \rangle$ , which can be both positive and negative. Headings will not be shifted beyond level 6 or below level 1. Instead, those headings will be shifted to level 6, when  $\langle shift\ amount \rangle$  is positive, and to level 1, when  $\langle shift\ amount \rangle$  is negative.

```
455 \@@_add_lua_option:nnn
456 { shiftHeadings }
457 { number }
458 { 0 }

459 defaultOptions.shiftHeadings = 0
```

`slice`= $\langle the\ beginning\ and\ the\ end\ of\ a\ slice \rangle$  default:  $\wedge$  \$

Two space-separated selectors that specify the slice of a document that will be processed, whereas the remainder of the document will be ignored. The following selectors are recognized:

- The circumflex ( $\wedge$ ) selects the beginning of a document.
- The dollar sign (\$) selects the end of a document.
- $\wedge\langle identifier \rangle$  selects the beginning of a section with the HTML attribute  $\# \langle identifier \rangle$  (see the `headerAttributes` option).
- $\$ \langle identifier \rangle$  selects the end of a section with the HTML attribute  $\# \langle identifier \rangle$ .
- $\langle identifier \rangle$  corresponds to  $\wedge\langle identifier \rangle$  for the first selector and to  $\$ \langle identifier \rangle$  for the second selector.

Specifying only a single selector,  $\langle identifier \rangle$ , is equivalent to specifying the two selectors  $\langle identifier \rangle \langle identifier \rangle$ , which is equivalent to  $\wedge\langle identifier \rangle \$ \langle identifier \rangle$ , i.e. the entire section with the HTML attribute  $\# \langle identifier \rangle$  will be selected.

```
460 \@@_add_lua_option:nnn
461 { slice }
462 { slice }
463 { \wedge ~$ }

464 defaultOptions.slice = " \wedge $"
```

`smartEllipses`=true, false default: false

- |                    |                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------|
| <code>true</code>  | Convert any ellipses in the input to the <code>\markdownRendererEllipsis</code> TeX macro. |
| <code>false</code> | Preserve all ellipses in the input.                                                        |

```

465 \@@_add_lua_option:nnn
466 { smartEllipses }
467 { boolean }
468 { false }
469 defaultOptions.smartEllipses = false

```

`startNumber=true, false`

default: true

- true**      Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererOliItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOliItem` T<sub>E</sub>X macro.

```

470 \@@_add_lua_option:nnn
471 { startNumber }
472 { boolean }
473 { true }
474 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

- true**      Enable the Pandoc strike-through syntax extension:

This ~~is deleted text.~~

- false**     Disable the Pandoc strike-through syntax extension.

```

475 \@@_add_lua_option:nnn
476 { strikeThrough }
477 { boolean }
478 { false }
479 defaultOptions.strikeThrough = false

```

`stripIndent=true, false`

default: false

- true**      Strip the minimal indentation of non-blank lines from all lines in a markdown document. Requires that the `preserveTabs` Lua option is disabled:

```

\documentclass{article}
\usepackage[stripIndent]{markdown}
\begin{document}
 \begin{markdown}
 Hello *world*!
 \end{markdown}
\end{document}

```

**false** Do not strip any indentation from the lines in a markdown document.

```

480 \@@_add_lua_option:nnn
481 { stripIndent }
482 { boolean }
483 { false }
484 defaultOptions.stripIndent = false

```

**subscripts=true, false** default: false

**true** Enable the Pandoc subscript syntax extension:

```
H~2~O is a liquid.
```

**false** Disable the Pandoc subscript syntax extension.

```

485 \@@_add_lua_option:nnn
486 { subscripts }
487 { boolean }
488 { false }
489 defaultOptions.subscripts = false

```

**superscripts=true, false** default: false

**true** Enable the Pandoc superscript syntax extension:

```
2^10^ is 1024.
```

**false** Disable the Pandoc superscript syntax extension.

```

490 \@@_add_lua_option:nnn
491 { superscripts }
492 { boolean }
493 { false }
494 defaultOptions.superscripts = false

```



`tableCaptions=true, false`

default: false

`true` Enable the Pandoc `table_captions` syntax extension for pipe tables (see the `pipeTables` option).

Right	Left	Default	Center
-----:	:-----	-----	:-----:
12	12	12	12
123	123	123	123
1	1	1	1
: Demonstration of pipe table syntax.			

`false` Disable the Pandoc `table_captions` syntax extension.

```
495 \@@_add_lua_option:nnn
496 { tableCaptions }
497 { boolean }
498 { false }
499 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: false

`true` Enable the Pandoc `task_lists` syntax extension.

- [ ] an unticked task list item
- [/] a half-checked task list item
- [X] a ticked task list item

`false` Disable the Pandoc `task_lists` syntax extension.

```
500 \@@_add_lua_option:nnn
501 { taskLists }
502 { boolean }
503 { false }
504 defaultOptions.taskLists = false
```

`texComments=true, false`

default: `false`

`true` Strip TeX-style comments.

```
\documentclass{article}
\usepackage[texComments]{markdown}
\begin{document}
\begin{markdown}
Hello *world*!
\end{markdown}
\end{document}
```

Always enabled when `hybrid` is enabled.

`false` Do not strip TeX-style comments.

```
505 \@@_add_lua_option:nnn
506 { texComments }
507 { boolean }
508 { false }
509 defaultOptions.texComments = false
```

`tightLists=true, false`

default: `true`

`true` Unordered and ordered lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renderers that may produce different output than lists that are not tight:

```
- This is
- a tight
- unordered list.

- This is

 not a tight

- unordered list.
```

`false` Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

```
510 \@@_add_lua_option:nnn
511 { tightLists }
512 { boolean }
513 { true }
```

```
514 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.

```
515 \@@_add_lua_option:nnn
516 { underscores }
517 { boolean }
518 { true }
519 \ExplSyntaxOff
520 defaultOptions.underscores = true
```

### 2.1.6 Command-Line Interface

The high-level operation of the Markdown package involves the communication between several programming layers: the plain  $\text{\TeX}$  layer hands markdown documents to the Lua layer. Lua converts the documents to  $\text{\TeX}$ , and hands the converted documents back to plain  $\text{\TeX}$  layer for typesetting, see Figure 2.

This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{\TeX}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{\TeX}$  engine includes a Lua interpreter, the package also requires shell access, which opens the door for a malicious actor to access the system. Last, but not least, the complexity of the procedure impedes debugging.

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to  $\text{\TeX}$  is also provided, see Figure 3.

```
521
522 local HELP_STRING = [[
523 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
524 where OPTIONS are documented in the Lua interface section of the
525 technical Markdown package documentation.
```

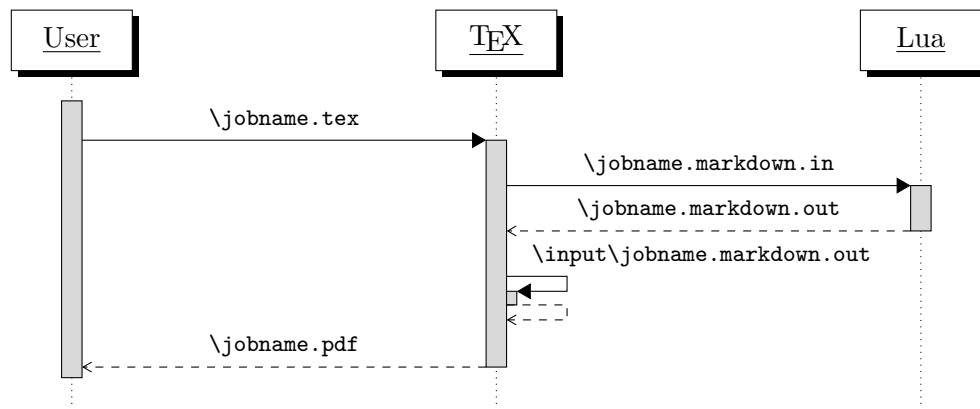


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface

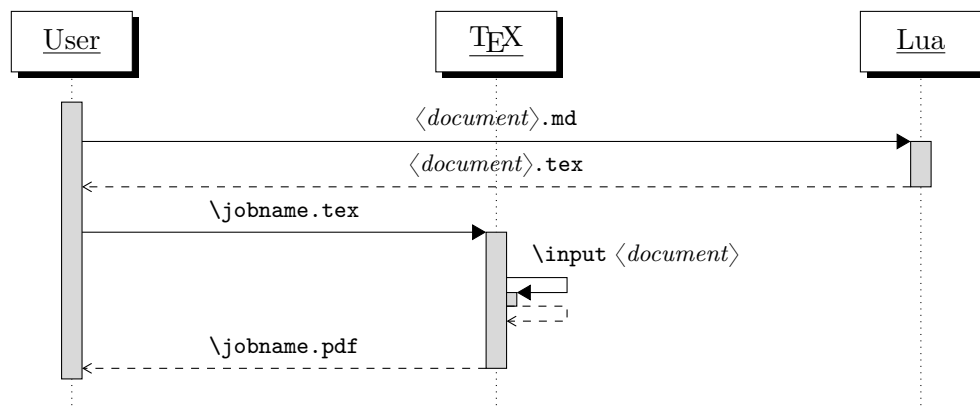


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

526
527 When OUTPUT_FILE is unspecified, the result of the conversion will be
528 written to the standard output. When INPUT_FILE is also unspecified, the
529 result of the conversion will be read from the standard input.
530
531 Report bugs to: witiko@mail.muni.cz
532 Markdown package home page: <https://github.com/witiko/markdown>]]
533
534 local VERSION_STRING = [[
535 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
536
537 Copyright (C)]] .. table.concat(metadata.copyright,
538 "\nCopyright (C) ") .. [[
539
540 License:]] .. metadata.license
541
542 local function warn(s)
543 io.stderr:write("Warning: " .. s .. "\n") end
544
545 local function error(s)
546 io.stderr:write("Error: " .. s .. "\n")
547 os.exit(1) end
548
549 local process_options = true
550 local options = {}
551 local input_filename
552 local output_filename
553 for i = 1, #arg do
554 if process_options then

```

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

```

555 if arg[i] == "--" then
556 process_options = false
557 goto continue

```

Unless the `--` argument has been specified before, an argument containing the equals sign (`=`) is assumed to be an option specification in a `<key>=<value>` format. The available options are listed in Section 2.1.3.

```

558 elseif arg[i]:match("=") then
559 local key, value = arg[i]:match("(.)=(.*)")

```

The `defaultOptions` table is consulted to identify whether `<value>` should be parsed as a string or as a boolean.

```

560 local default_type = type(defaultOptions[key])
561 if default_type == "boolean" then

```

```

562 options[key] = (value == "true")
563 elseif default_type == "number" then
564 options[key] = tonumber(value)
565 elseif default_type == "table" then
566 options[key] = {}
567 for item in value:gmatch("[^,]+") do
568 table.insert(options[key], item)
569 end
570 else
571 if default_type ~= "string" then
572 if default_type == "nil" then
573 warn('Option "' .. key .. '" not recognized.')
574 else
575 warn('Option "' .. key .. '" type not recognized, please file ' ..
576 'a report to the package maintainer.')
577 end
578 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
579 key .. '" as a string.')
580 end
581 options[key] = value
582 end
583 goto continue

```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

```

584 elseif arg[i] == "--help" or arg[i] == "-h" then
585 print(HELP_STRING)
586 os.exit()

```

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

```

587 elseif arg[i] == "--version" or arg[i] == "-v" then
588 print(VERSION_STRING)
589 os.exit()
590 end
591 end

```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a  $\text{T}_{\text{E}}\text{X}$  document.

```

592 if input_filename == nil then
593 input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the  $\text{T}_{\text{E}}\text{X}$  document that will result from the conversion.

```

594 elseif output_filename == nil then
595 output_filename = arg[i]
596 else
597 error('Unexpected argument: "' .. arg[i] .. '".')
598 end
599 ::continue::
600 end

```

The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```
texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex
```

to convert the Markdown document `hello.md` to a T<sub>E</sub>X document `hello.tex`. After the Markdown package for our T<sub>E</sub>X format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

## 2.2 Plain T<sub>E</sub>X Interface

The plain T<sub>E</sub>X interface provides macros for the typesetting of markdown input from within plain T<sub>E</sub>X, for setting the Lua interface options (see Section 2.1.3) used during the conversion from markdown to plain T<sub>E</sub>X and for changing the way markdown the tokens are rendered.

```

601 \def\markdownLastModified{((LASTMODIFIED))}%
602 \def\markdownVersion{((VERSION))}%

```

The plain T<sub>E</sub>X interface is implemented by the `markdown.tex` file that can be loaded as follows:

```
\input markdown
```

It is expected that the special plain T<sub>E</sub>X 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.

```

603 \let\markdownBegin\relax
604 \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<sub>E</sub>X [4, 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<sub>E</sub>X 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 T<sub>E</sub>X 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 T<sub>E</sub>X.

605 \let\markdownInput\relax

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

The following example plain T<sub>E</sub>X code showcases the usage of the `\markdownInput` macro:



```

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

```

## 2.2.2 Options

The plain  $\text{\TeX}$  options are represented by  $\text{\TeX}$  commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain  $\text{\TeX}$  interface.

To enable the enumeration of plain  $\text{\TeX}$  options, we will maintain the `\g_@@_plain_tex_options_seq` sequence.

```

606 \ExplSyntaxOn
607 \seq_new:N \g_@@_plain_tex_options_seq

```

To enable the reflection of default plain  $\text{\TeX}$  options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop` and `\g_@@_plain_tex_option_types_prop` property lists, respectively.

```

608 \prop_new:N \g_@@_plain_tex_option_types_prop
609 \prop_new:N \g_@@_default_plain_tex_options_prop
610 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
611 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
612 \cs_new:Nn
613 \@@_add_plain_tex_option:nnn
614 {
615 \@@_add_option:Vnnn
616 \c_@@_option_layer_plain_tex_tl
617 { #1 }
618 { #2 }
619 { #3 }
620 }

```

**2.2.2.1 Finalizing and Freezing the Cache** The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `\markdownOptionFrozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain  $\text{\TeX}$  document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `\markdownOptionFinalizeCache` option, and uses it to typeset the plain  $\text{\TeX}$  document without invoking Lua. As a result, the plain  $\text{\TeX}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

```

621 \@@_add_plain_tex_option:nnn
622 { frozenCache }

```

```

623 { boolean }
624 { false }

```

The standard usage of the above two options is as follows:

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `\markdownOptionFinalizeCache` option.
4. Typeset the plain T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

**2.2.2.2 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 T<sub>E</sub>X in T<sub>E</sub>X 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 T<sub>E</sub>X engines tend to put quotation marks around `\jobname`, when it contains spaces.

```

625 \@@_add_plain_tex_option:nnn
626 { helperScriptFileName }
627 { path }
628 { \jobname.markdown.lua }

```

The `\markdownOptionHelperScriptFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the helper Lua script file, use the `\g_luabridge_helper_script_filename_str` macro from the `lt3luabridge` package.

```

629 \str_new:N
630 \g_luabridge_helper_script_filename_str
631 \tl_gset:Nn
632 \g_luabridge_helper_script_filename_str
633 { \markdownOptionHelperScriptFileName }

```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```

634 \@@_add_plain_tex_option:nnn
635 { inputTempFileName }
636 { path }
637 { \jobname.markdown.in }

```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain T<sub>E</sub>X in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same

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

```
638 \@@_add_plain_tex_option:nnn
639 { outputTempFileName }
640 { path }
641 { \jobname.markdown.out }
```

The `\markdownOptionOutputTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0.

```
642 \str_new:N
643 \g_luabridge_standard_output_filename_str
644 \tl_gset:Nn
645 \g_luabridge_standard_output_filename_str
646 { \markdownOptionOutputTempFileName }
```

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 T<sub>E</sub>X in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
647 \@@_add_plain_tex_option:nnn
648 { errorTempFileName }
649 { path }
650 { \jobname.markdown.err }
```

The `\markdownOptionErrorTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the temporary file for Lua errors, use the `\g_luabridge_error_output_filename_str` macro from the `lt3luabridge` package.

```
651 \str_new:N
652 \g_luabridge_error_output_filename_str
653 \tl_gset:Nn
654 \g_luabridge_error_output_filename_str
655 { \markdownOptionErrorTempFileName }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain T<sub>E</sub>X implementation. The option defaults to `..`.

The path must be set to the same value as the `-output-directory` option of your T<sub>E</sub>X engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
656 \@@_add_plain_tex_option:nnn
657 { outputDir }
658 { path }
659 { . }
```

Here, we automatically define plain TeX macros for the above plain TeX options.

Furthermore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain TeX implementation, only passed along to Lua.

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.

```

660 \cs_new:Nn \@@_plain_tex_define_option_commands:
661 {
662 \seq_map_inline:Nn
663 \g_@@_option_layers_seq
664 {
665 \seq_map_inline:cn
666 { g_@@_ ##1 _options_seq }
667 {
668 \@@_plain_tex_define_option_command:n
669 { ####1 }
670 }
671 }
672 }
673 \cs_new:Nn \@@_plain_tex_define_option_command:n
674 {
675 \@@_get_default_option_value:nN
676 { #1 }
677 \l_tmpa_tl
678 \@@_set_option_value:nV
679 { #1 }
680 \l_tmpa_tl
681 }
682 \cs_new:Nn
683 \@@_set_option_value:nn
684 {
685 \@@_define_option:n
686 { #1 }
687 \@@_get_option_type:nN
688 { #1 }
689 \l_tmpa_tl
690 \str_if_eq:NNTF
691 \c_@@_option_type_counter_tl
692 \l_tmpa_tl
693 {
694 \@@_option_tl_to_csname:nN
695 { #1 }
696 \l_tmpa_tl
697 \int_gset:cn

```

```

698 { \l_tmpa_tl }
699 { #2 }
700 }
701 {
702 \@@_option_tl_to_csname:nN
703 { #1 }
704 \l_tmpa_tl
705 \cs_set:cpn
706 { \l_tmpa_tl }
707 { #2 }
708 }
709 }
710 \cs_generate_variant:Nn
711 \@@_set_option_value:nn
712 { nV }
713 \cs_new:Nn
714 \@@_define_option:n
715 {
716 \@@_option_tl_to_csname:nN
717 { #1 }
718 \l_tmpa_tl
719 \cs_if_free:cT
720 { \l_tmpa_tl }
721 {
722 \@@_get_option_type:nN
723 { #1 }
724 \l_tmpb_tl
725 \str_if_eq:NNT
726 \c_@@_option_type_counter_tl
727 \l_tmpb_tl
728 {
729 \@@_option_tl_to_csname:nN
730 { #1 }
731 \l_tmpa_tl
732 \int_new:c
733 { \l_tmpa_tl }
734 }
735 }
736 }
737 \@@_plain_tex_define_option_commands:

```

**2.2.2.3 Miscellaneous Options** The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [5]

or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

738 \seq_put_right:Nn
739 \g_@@_plain_tex_options_seq
740 { stripPercentSigns }
741 \prop_put:Nnn
742 \g_@@_plain_tex_option_types_prop
743 { stripPercentSigns }
744 { boolean }
745 \prop_put:Nnx
746 \g_@@_default_plain_tex_options_prop
747 { stripPercentSigns }
748 { false }
749 \ExplSyntaxOff

```

### 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).

To enable the enumeration of token renderers, we will maintain the `\g_@@_renderers_seq` sequence.

```

750 \ExplSyntaxOn
751 \seq_new:N \g_@@_renderers_seq

```

To enable the reflection of token renderers and their parameters, we will maintain the `\g_@@_renderer_arities_prop` property list.

```

752 \prop_new:N \g_@@_renderer_arities_prop
753 \ExplSyntaxOff

```

**2.2.3.1 Tickbox Renderers** The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⌚, U+231B) or Ballot Box (☐, U+2610) Unicode characters are encountered in the markdown input, respectively.

```

754 \def\markdownRendererTickedBox{%
755 \markdownRendererTickedBoxPrototype}%
756 \ExplSyntaxOn
757 \seq_put_right:Nn
758 \g_@@_renderers_seq
759 { tickedBox }
760 \prop_put:Nnn
761 \g_@@_renderer_arities_prop

```

```

762 { tickedTextBox }
763 { 0 }
764 \ExplSyntaxOff
765 \def\markdownRendererHalfTickedTextBox{%
766 \markdownRendererHalfTickedTextBoxPrototype}%
767 \ExplSyntaxOn
768 \seq_put_right:Nn
769 \g_@@_renderers_seq
770 { halfTickedTextBox }
771 \prop_put:Nnn
772 \g_@@_renderer_arities_prop
773 { halfTickedTextBox }
774 { 0 }
775 \ExplSyntaxOff
776 \def\markdownRendererUntickedTextBox{%
777 \markdownRendererUntickedTextBoxPrototype}%
778 \ExplSyntaxOn
779 \seq_put_right:Nn
780 \g_@@_renderers_seq
781 { untickedTextBox }
782 \prop_put:Nnn
783 \g_@@_renderer_arities_prop
784 { untickedTextBox }
785 { 0 }
786 \ExplSyntaxOff

```

**2.2.3.2 Markdown Document Renderers** The `\markdownRendererDocumentBegin` and `\markdownRendererDocumentEnd` macros represent the beginning and the end of a *markdown* document. The macros receive no arguments.

A  $\text{\TeX}$  document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```

787 \def\markdownRendererDocumentBegin{%
788 \markdownRendererDocumentBeginPrototype}%
789 \ExplSyntaxOn
790 \seq_put_right:Nn
791 \g_@@_renderers_seq
792 { documentBegin }
793 \prop_put:Nnn
794 \g_@@_renderer_arities_prop
795 { documentBegin }
796 { 0 }
797 \ExplSyntaxOff
798 \def\markdownRendererDocumentEnd{%

```

```

799 \markdownRendererDocumentEndPrototype}%
800 \ExplSyntaxOn
801 \seq_put_right:Nn
802 \g_@@_renderers_seq
803 { documentEnd }
804 \prop_put:Nnn
805 \g_@@_renderer_arities_prop
806 { documentEnd }
807 { 0 }
808 \ExplSyntaxOff

```

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

```

809 \def\markdownRendererInterblockSeparator{%
810 \markdownRendererInterblockSeparatorPrototype}%
811 \ExplSyntaxOn
812 \seq_put_right:Nn
813 \g_@@_renderers_seq
814 { interblockSeparator }
815 \prop_put:Nnn
816 \g_@@_renderer_arities_prop
817 { interblockSeparator }
818 { 0 }
819 \ExplSyntaxOff

```

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

```

820 \def\markdownRendererLineBreak{%
821 \markdownRendererLineBreakPrototype}%
822 \ExplSyntaxOn
823 \seq_put_right:Nn
824 \g_@@_renderers_seq
825 { lineBreak }
826 \prop_put:Nnn
827 \g_@@_renderer_arities_prop
828 { lineBreak }
829 { 0 }
830 \ExplSyntaxOff

```

**2.2.3.5 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 enabled. The macro receives no arguments.

```

831 \def\markdownRendererEllipsis{%

```



```

832 \markdownRendererEllipsisPrototype}%
833 \ExplSyntaxOn
834 \seq_put_right:Nn
835 \g_@@_renderers_seq
836 { ellipsis }
837 \prop_put:Nnn
838 \g_@@_renderer_arities_prop
839 { ellipsis }
840 { 0 }
841 \ExplSyntaxOff

```

**2.2.3.6 Non-Breaking Space Renderer** The `\markdownRendererNbsp` macro represents a non-breaking space.

```

842 \def\markdownRendererNbsp{%
843 \markdownRendererNbspPrototype}%
844 \ExplSyntaxOn
845 \seq_put_right:Nn
846 \g_@@_renderers_seq
847 { nbsp }
848 \prop_put:Nnn
849 \g_@@_renderer_arities_prop
850 { nbsp }
851 { 0 }
852 \ExplSyntaxOff

```

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

```

853 \def\markdownRendererLeftBrace{%
854 \markdownRendererLeftBracePrototype}%
855 \ExplSyntaxOn
856 \seq_put_right:Nn
857 \g_@@_renderers_seq
858 { leftBrace }
859 \prop_put:Nnn
860 \g_@@_renderer_arities_prop
861 { leftBrace }
862 { 0 }
863 \ExplSyntaxOff
864 \def\markdownRendererRightBrace{%
865 \markdownRendererRightBracePrototype}%
866 \ExplSyntaxOn
867 \seq_put_right:Nn
868 \g_@@_renderers_seq
869 { rightBrace }

```

```

870 \prop_put:Nnn
871 \g_@@_renderer_arities_prop
872 { rightBrace }
873 { 0 }
874 \ExplSyntaxOff
875 \def\markdownRendererDollarSign{%
876 \markdownRendererDollarSignPrototype}%
877 \ExplSyntaxOn
878 \seq_put_right:Nn
879 \g_@@_renderers_seq
880 { dollarSign }
881 \prop_put:Nnn
882 \g_@@_renderer_arities_prop
883 { dollarSign }
884 { 0 }
885 \ExplSyntaxOff
886 \def\markdownRendererPercentSign{%
887 \markdownRendererPercentSignPrototype}%
888 \ExplSyntaxOn
889 \seq_put_right:Nn
890 \g_@@_renderers_seq
891 { percentSign }
892 \prop_put:Nnn
893 \g_@@_renderer_arities_prop
894 { percentSign }
895 { 0 }
896 \ExplSyntaxOff
897 \def\markdownRendererAmpersand{%
898 \markdownRendererAmpersandPrototype}%
899 \ExplSyntaxOn
900 \seq_put_right:Nn
901 \g_@@_renderers_seq
902 { ampersand }
903 \prop_put:Nnn
904 \g_@@_renderer_arities_prop
905 { ampersand }
906 { 0 }
907 \ExplSyntaxOff
908 \def\markdownRendererUnderscore{%
909 \markdownRendererUnderscorePrototype}%
910 \ExplSyntaxOn
911 \seq_put_right:Nn
912 \g_@@_renderers_seq
913 { underscore }
914 \prop_put:Nnn
915 \g_@@_renderer_arities_prop
916 { underscore }

```

```

917 { 0 }
918 \ExplSyntaxOff
919 \def\markdownRendererHash{%
920 \markdownRendererHashPrototype}%
921 \ExplSyntaxOn
922 \seq_put_right:Nn
923 \g_@@_renderers_seq
924 { hash }
925 \prop_put:Nnn
926 \g_@@_renderer_arities_prop
927 { hash }
928 { 0 }
929 \ExplSyntaxOff
930 \def\markdownRendererCircumflex{%
931 \markdownRendererCircumflexPrototype}%
932 \ExplSyntaxOn
933 \seq_put_right:Nn
934 \g_@@_renderers_seq
935 { circumflex }
936 \prop_put:Nnn
937 \g_@@_renderer_arities_prop
938 { circumflex }
939 { 0 }
940 \ExplSyntaxOff
941 \def\markdownRendererBackslash{%
942 \markdownRendererBackslashPrototype}%
943 \ExplSyntaxOn
944 \seq_put_right:Nn
945 \g_@@_renderers_seq
946 { backslash }
947 \prop_put:Nnn
948 \g_@@_renderer_arities_prop
949 { backslash }
950 { 0 }
951 \ExplSyntaxOff
952 \def\markdownRendererTilde{%
953 \markdownRendererTildePrototype}%
954 \ExplSyntaxOn
955 \seq_put_right:Nn
956 \g_@@_renderers_seq
957 { tilde }
958 \prop_put:Nnn
959 \g_@@_renderer_arities_prop
960 { tilde }
961 { 0 }
962 \ExplSyntaxOff
963 \def\markdownRendererPipe{%

```

```

964 \markdownRendererPipePrototype}%
965 \ExplSyntaxOn
966 \seq_put_right:Nn
967 \g_@@_renderers_seq
968 { pipe }
969 \prop_put:Nnn
970 \g_@@_renderer_arities_prop
971 { pipe }
972 { 0 }
973 \ExplSyntaxOff

```

**2.2.3.8 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.

```

974 \def\markdownRendererCodeSpan{%
975 \markdownRendererCodeSpanPrototype}%
976 \ExplSyntaxOn
977 \seq_put_right:Nn
978 \g_@@_renderers_seq
979 { codeSpan }
980 \prop_put:Nnn
981 \g_@@_renderer_arities_prop
982 { codeSpan }
983 { 1 }
984 \ExplSyntaxOff

```

**2.2.3.9 Link Renderer** The `\markdownRendererLink` macro represents a hyperlink. 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.

```

985 \def\markdownRendererLink{%
986 \markdownRendererLinkPrototype}%
987 \ExplSyntaxOn
988 \seq_put_right:Nn
989 \g_@@_renderers_seq
990 { link }
991 \prop_put:Nnn
992 \g_@@_renderer_arities_prop
993 { link }
994 { 4 }
995 \ExplSyntaxOff

```

**2.2.3.10 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.

```

996 \def\markdownRendererImage{%
997 \markdownRendererImagePrototype}%
998 \ExplSyntaxOn
999 \seq_put_right:Nn
1000 \g_@@_renderers_seq
1001 { image }
1002 \prop_put:Nnn
1003 \g_@@_renderer_arities_prop
1004 { image }
1005 { 4 }
1006 \ExplSyntaxOff

```

**2.2.3.11 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.

```

1007 \def\markdownRendererContentBlock{%
1008 \markdownRendererContentBlockPrototype}%
1009 \ExplSyntaxOn
1010 \seq_put_right:Nn
1011 \g_@@_renderers_seq
1012 { contentBlock }
1013 \prop_put:Nnn
1014 \g_@@_renderer_arities_prop
1015 { contentBlock }
1016 { 4 }
1017 \ExplSyntaxOff

```

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

```

1018 \def\markdownRendererContentBlockOnlineImage{%
1019 \markdownRendererContentBlockOnlineImagePrototype}%
1020 \ExplSyntaxOn
1021 \seq_put_right:Nn
1022 \g_@@_renderers_seq
1023 { contentBlockOnlineImage }
1024 \prop_put:Nnn
1025 \g_@@_renderer_arities_prop
1026 { contentBlockOnlineImage }
1027 { 4 }
1028 \ExplSyntaxOff

```

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<sup>7</sup> 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 T<sub>E</sub>X 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 Sotkov [3] is a good starting point.

```

1029 \def\markdownRendererContentBlockCode{%
1030 \markdownRendererContentBlockCodePrototype}%
1031 \ExplSyntaxOn
1032 \seq_put_right:Nn
1033 \g_@@_renderers_seq
1034 { contentBlockCode }
1035 \prop_put:Nnn
1036 \g_@@_renderer_arities_prop
1037 { contentBlockCode }
1038 { 5 }
1039 \ExplSyntaxOff

```

**2.2.3.12 Bullet List Renderers** The `\markdownRendererUllBegin` 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.

```

1040 \def\markdownRendererUllBegin{%
1041 \markdownRendererUllBeginPrototype}%
1042 \ExplSyntaxOn
1043 \seq_put_right:Nn
1044 \g_@@_renderers_seq
1045 { ullBegin }
1046 \prop_put:Nnn
1047 \g_@@_renderer_arities_prop
1048 { ullBegin }
1049 { 0 }
1050 \ExplSyntaxOff

```

---

<sup>7</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

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 disabled. The macro receives no arguments.

```

1051 \def\markdownRendererUlBeginTight{%
1052 \markdownRendererUlBeginTightPrototype}%
1053 \ExplSyntaxOn
1054 \seq_put_right:Nn
1055 \g_@@_renderers_seq
1056 { ulBeginTight }
1057 \prop_put:Nnn
1058 \g_@@_renderer_arities_prop
1059 { ulBeginTight }
1060 { 0 }
1061 \ExplSyntaxOff

```

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

```

1062 \def\markdownRendererUlItem{%
1063 \markdownRendererUlItemPrototype}%
1064 \ExplSyntaxOn
1065 \seq_put_right:Nn
1066 \g_@@_renderers_seq
1067 { ulItem }
1068 \prop_put:Nnn
1069 \g_@@_renderer_arities_prop
1070 { ulItem }
1071 { 0 }
1072 \ExplSyntaxOff

```

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

```

1073 \def\markdownRendererUlItemEnd{%
1074 \markdownRendererUlItemEndPrototype}%
1075 \ExplSyntaxOn
1076 \seq_put_right:Nn
1077 \g_@@_renderers_seq
1078 { ulItemEnd }
1079 \prop_put:Nnn
1080 \g_@@_renderer_arities_prop
1081 { ulItemEnd }
1082 { 0 }
1083 \ExplSyntaxOff

```

The `\markdownRendererUEnd` 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.

```

1084 \def\markdownRendererUEnd{%
1085 \markdownRendererUEndPrototype}%
1086 \ExplSyntaxOn
1087 \seq_put_right:Nn
1088 \g_@@_renderers_seq
1089 { ulEnd }
1090 \prop_put:Nnn
1091 \g_@@_renderer_arities_prop
1092 { ulEnd }
1093 { 0 }
1094 \ExplSyntaxOff

```

The `\markdownRendererUEndTight` 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 disabled. The macro receives no arguments.

```

1095 \def\markdownRendererUEndTight{%
1096 \markdownRendererUEndTightPrototype}%
1097 \ExplSyntaxOn
1098 \seq_put_right:Nn
1099 \g_@@_renderers_seq
1100 { ulEndTight }
1101 \prop_put:Nnn
1102 \g_@@_renderer_arities_prop
1103 { ulEndTight }
1104 { 0 }
1105 \ExplSyntaxOff

```

**2.2.3.13 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). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1106 \def\markdownRendererOlBegin{%
1107 \markdownRendererOlBeginPrototype}%
1108 \ExplSyntaxOn
1109 \seq_put_right:Nn
1110 \g_@@_renderers_seq
1111 { olBegin }
1112 \prop_put:Nnn
1113 \g_@@_renderer_arities_prop
1114 { olBegin }
1115 { 0 }

```



1116 \ExplSyntaxOff

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 enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```
1117 \def\markdownRendererOlBeginTight{%
1118 \markdownRendererOlBeginTightPrototype}%
1119 \ExplSyntaxOn
1120 \seq_put_right:Nn
1121 \g_@@_renderers_seq
1122 { olBeginTight }
1123 \prop_put:Nnn
1124 \g_@@_renderer_arities_prop
1125 { olBeginTight }
1126 { 0 }
1127 \ExplSyntaxOff
```

The `\markdownRendererFancyOlBegin` macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives two arguments: the style of the list item labels (`Decimal`, `LowerRoman`, `UpperRoman`, `LowerAlpha`, and `UpperAlpha`), and the style of delimiters between list item labels and texts (`Default`, `OneParen`, and `Period`).

```
1128 \def\markdownRendererFancyOlBegin{%
1129 \markdownRendererFancyOlBeginPrototype}%
1130 \ExplSyntaxOn
1131 \seq_put_right:Nn
1132 \g_@@_renderers_seq
1133 { fancyOlBegin }
1134 \prop_put:Nnn
1135 \g_@@_renderer_arities_prop
1136 { fancyOlBegin }
1137 { 2 }
1138 \ExplSyntaxOff
```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```
1139 \def\markdownRendererFancyOlBeginTight{%
1140 \markdownRendererFancyOlBeginTightPrototype}%
1141 \ExplSyntaxOn
```

```

1142 \seq_put_right:Nn
1143 \g_@@_renderers_seq
1144 { fancyOlBeginTight }
1145 \prop_put:Nnn
1146 \g_@@_renderer_arities_prop
1147 { fancyOlBeginTight }
1148 { 2 }
1149 \ExplSyntaxOff

```

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

```

1150 \def\markdownRendererOlItem{%
1151 \markdownRendererOlItemPrototype}%
1152 \ExplSyntaxOn
1153 \seq_put_right:Nn
1154 \g_@@_renderers_seq
1155 { olItem }
1156 \prop_put:Nnn
1157 \g_@@_renderer_arities_prop
1158 { olItem }
1159 { 0 }
1160 \ExplSyntaxOff

```

The `\markdownRendererOlItemEnd` macro represents the end of an item in an ordered list. This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1161 \def\markdownRendererOlItemEnd{%
1162 \markdownRendererOlItemEndPrototype}%
1163 \ExplSyntaxOn
1164 \seq_put_right:Nn
1165 \g_@@_renderers_seq
1166 { olItemEnd }
1167 \prop_put:Nnn
1168 \g_@@_renderer_arities_prop
1169 { olItemEnd }
1170 { 0 }
1171 \ExplSyntaxOff

```

The `\markdownRendererOlItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is enabled and the `fancyLists` option is disabled. The macro receives a single numeric argument that corresponds to the item number.

```

1172 \def\markdownRendererOlItemWithNumber{%
1173 \markdownRendererOlItemWithNumberPrototype}%
1174 \ExplSyntaxOn

```

```

1175 \seq_put_right:Nn
1176 \g_@@_renderers_seq
1177 { olItemWithNumber }
1178 \prop_put:Nnn
1179 \g_@@_renderer_arities_prop
1180 { olItemWithNumber }
1181 { 1 }
1182 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItem` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```

1183 \def\markdownRendererFancyOlItem{%
1184 \markdownRendererFancyOlItemPrototype}%
1185 \ExplSyntaxOn
1186 \seq_put_right:Nn
1187 \g_@@_renderers_seq
1188 { fancyOlItem }
1189 \prop_put:Nnn
1190 \g_@@_renderer_arities_prop
1191 { fancyOlItem }
1192 { 0 }
1193 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

1194 \def\markdownRendererFancyOlItemEnd{%
1195 \markdownRendererFancyOlItemEndPrototype}%
1196 \ExplSyntaxOn
1197 \seq_put_right:Nn
1198 \g_@@_renderers_seq
1199 { fancyOlItemEnd }
1200 \prop_put:Nnn
1201 \g_@@_renderer_arities_prop
1202 { fancyOlItemEnd }
1203 { 0 }
1204 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and `fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```

1205 \def\markdownRendererFancyOlItemWithNumber{%
1206 \markdownRendererFancyOlItemWithNumberPrototype}%
1207 \ExplSyntaxOn

```

```

1208 \seq_put_right:Nn
1209 \g_@@_renderers_seq
1210 { fancyO1ItemWithNumber }
1211 \prop_put:Nnn
1212 \g_@@_renderer_arities_prop
1213 { fancyO1ItemWithNumber }
1214 { 1 }
1215 \ExplSyntaxOff

```

The `\markdownRendererO1End` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1216 \def\markdownRendererO1End{%
1217 \markdownRendererO1EndPrototype}%
1218 \ExplSyntaxOn
1219 \seq_put_right:Nn
1220 \g_@@_renderers_seq
1221 { olEnd }
1222 \prop_put:Nnn
1223 \g_@@_renderer_arities_prop
1224 { olEnd }
1225 { 0 }
1226 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` 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 enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

1227 \def\markdownRendererO1EndTight{%
1228 \markdownRendererO1EndTightPrototype}%
1229 \ExplSyntaxOn
1230 \seq_put_right:Nn
1231 \g_@@_renderers_seq
1232 { olEndTight }
1233 \prop_put:Nnn
1234 \g_@@_renderer_arities_prop
1235 { olEndTight }
1236 { 0 }
1237 \ExplSyntaxOff

```

The `\markdownRendererFancyO1End` macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

1238 \def\markdownRendererFancyO1End{%

```

```

1239 \markdownRendererFancy0lEndPrototype}%
1240 \ExplSyntaxOn
1241 \seq_put_right:Nn
1242 \g_@@_renderers_seq
1243 { fancy0lEnd }
1244 \prop_put:Nnn
1245 \g_@@_renderer_arities_prop
1246 { fancy0lEnd }
1247 { 0 }
1248 \ExplSyntaxOff

```

The `\markdownRendererFancy0lEndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

1249 \def\markdownRendererFancy0lEndTight{%
1250 \markdownRendererFancy0lEndTightPrototype}%
1251 \ExplSyntaxOn
1252 \seq_put_right:Nn
1253 \g_@@_renderers_seq
1254 { fancy0lEndTight }
1255 \prop_put:Nnn
1256 \g_@@_renderer_arities_prop
1257 { fancy0lEndTight }
1258 { 0 }
1259 \ExplSyntaxOff

```

**2.2.3.14 Definition List Renderers** The following macros are only produced, when the `definitionLists` option is enabled.

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.

```

1260 \def\markdownRendererDlBegin{%
1261 \markdownRendererDlBeginPrototype}%
1262 \ExplSyntaxOn
1263 \seq_put_right:Nn
1264 \g_@@_renderers_seq
1265 { dlBegin }
1266 \prop_put:Nnn
1267 \g_@@_renderer_arities_prop
1268 { dlBegin }
1269 { 0 }
1270 \ExplSyntaxOff

```

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 disabled. The macro receives no arguments.

```

1271 \def\markdownRendererDlBeginTight{%
1272 \markdownRendererDlBeginTightPrototype}%
1273 \ExplSyntaxOn
1274 \seq_put_right:Nn
1275 \g_@@_renderers_seq
1276 { dlBeginTight }
1277 \prop_put:Nnn
1278 \g_@@_renderer_arities_prop
1279 { dlBeginTight }
1280 { 0 }
1281 \ExplSyntaxOff

```

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

```

1282 \def\markdownRendererDlItem{%
1283 \markdownRendererDlItemPrototype}%
1284 \ExplSyntaxOn
1285 \seq_put_right:Nn
1286 \g_@@_renderers_seq
1287 { dlItem }
1288 \prop_put:Nnn
1289 \g_@@_renderer_arities_prop
1290 { dlItem }
1291 { 1 }
1292 \ExplSyntaxOff

```

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

```

1293 \def\markdownRendererDlItemEnd{%
1294 \markdownRendererDlItemEndPrototype}%
1295 \ExplSyntaxOn
1296 \seq_put_right:Nn
1297 \g_@@_renderers_seq
1298 { dlItemEnd }
1299 \prop_put:Nnn
1300 \g_@@_renderer_arities_prop
1301 { dlItemEnd }
1302 { 0 }
1303 \ExplSyntaxOff

```

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

```

1304 \def\markdownRendererDlDefinitionBegin{%
1305 \markdownRendererDlDefinitionBeginPrototype}%

```

```

1306 \ExplSyntaxOn
1307 \seq_put_right:Nn
1308 \g_@@_renderers_seq
1309 { dlDefinitionBegin }
1310 \prop_put:Nnn
1311 \g_@@_renderer_arities_prop
1312 { dlDefinitionBegin }
1313 { 0 }
1314 \ExplSyntaxOff

```

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

```

1315 \def\markdownRendererDlDefinitionEnd{%
1316 \markdownRendererDlDefinitionEndPrototype}%
1317 \ExplSyntaxOn
1318 \seq_put_right:Nn
1319 \g_@@_renderers_seq
1320 { dlDefinitionEnd }
1321 \prop_put:Nnn
1322 \g_@@_renderer_arities_prop
1323 { dlDefinitionEnd }
1324 { 0 }
1325 \ExplSyntaxOff

```

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.

```

1326 \def\markdownRendererDlEnd{%
1327 \markdownRendererDlEndPrototype}%
1328 \ExplSyntaxOn
1329 \seq_put_right:Nn
1330 \g_@@_renderers_seq
1331 { dlEnd }
1332 \prop_put:Nnn
1333 \g_@@_renderer_arities_prop
1334 { dlEnd }
1335 { 0 }
1336 \ExplSyntaxOff

```

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 disabled. The macro receives no arguments.

```

1337 \def\markdownRendererDlEndTight{%
1338 \markdownRendererDlEndTightPrototype}%
1339 \ExplSyntaxOn

```

```

1340 \seq_put_right:Nn
1341 \g_@@_renderers_seq
1342 { dlEndTight }
1343 \prop_put:Nnn
1344 \g_@@_renderer_arities_prop
1345 { dlEndTight }
1346 { 0 }
1347 \ExplSyntaxOff

```

**2.2.3.15 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.

```

1348 \def\markdownRendererEmphasis{%
1349 \markdownRendererEmphasisPrototype}%
1350 \ExplSyntaxOn
1351 \seq_put_right:Nn
1352 \g_@@_renderers_seq
1353 { emphasis }
1354 \prop_put:Nnn
1355 \g_@@_renderer_arities_prop
1356 { emphasis }
1357 { 1 }
1358 \ExplSyntaxOff

```

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

```

1359 \def\markdownRendererStrongEmphasis{%
1360 \markdownRendererStrongEmphasisPrototype}%
1361 \ExplSyntaxOn
1362 \seq_put_right:Nn
1363 \g_@@_renderers_seq
1364 { strongEmphasis }
1365 \prop_put:Nnn
1366 \g_@@_renderer_arities_prop
1367 { strongEmphasis }
1368 { 1 }
1369 \ExplSyntaxOff

```

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

```

1370 \def\markdownRendererBlockQuoteBegin{%
1371 \markdownRendererBlockQuoteBeginPrototype}%
1372 \ExplSyntaxOn
1373 \seq_put_right:Nn

```



```

1374 \g_@@_renderers_seq
1375 { blockQuoteBegin }
1376 \prop_put:Nnn
1377 \g_@@_renderer_arities_prop
1378 { blockQuoteBegin }
1379 { 0 }
1380 \ExplSyntaxOff

```

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

```

1381 \def\markdownRendererBlockQuoteEnd{%
1382 \markdownRendererBlockQuoteEndPrototype}%
1383 \ExplSyntaxOn
1384 \seq_put_right:Nn
1385 \g_@@_renderers_seq
1386 { blockQuoteEnd }
1387 \prop_put:Nnn
1388 \g_@@_renderer_arities_prop
1389 { blockQuoteEnd }
1390 { 0 }
1391 \ExplSyntaxOff

```

**2.2.3.17 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.

```

1392 \def\markdownRendererInputVerbatim{%
1393 \markdownRendererInputVerbatimPrototype}%
1394 \ExplSyntaxOn
1395 \seq_put_right:Nn
1396 \g_@@_renderers_seq
1397 { inputVerbatim }
1398 \prop_put:Nnn
1399 \g_@@_renderer_arities_prop
1400 { inputVerbatim }
1401 { 1 }
1402 \ExplSyntaxOff

```

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

```

1403 \def\markdownRendererInputFencedCode{%
1404 \markdownRendererInputFencedCodePrototype}%
1405 \ExplSyntaxOn
1406 \seq_put_right:Nn

```

```

1407 \g_@@_renderers_seq
1408 { inputFencedCode }
1409 \prop_put:Nnn
1410 \g_@@_renderer_arities_prop
1411 { inputFencedCode }
1412 { 2 }
1413 \ExplSyntaxOff

```

**2.2.3.18 YAML Metadata Renderers** The `\markdownRendererJekyllDataBegin` macro represents the beginning of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1414 \def\markdownRendererJekyllDataBegin{%
1415 \markdownRendererJekyllDataBeginPrototype}%
1416 \ExplSyntaxOn
1417 \seq_put_right:Nn
1418 \g_@@_renderers_seq
1419 { jekyllDataBegin }
1420 \prop_put:Nnn
1421 \g_@@_renderer_arities_prop
1422 { jekyllDataBegin }
1423 { 0 }
1424 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEnd` macro represents the end of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1425 \def\markdownRendererJekyllDataEnd{%
1426 \markdownRendererJekyllDataEndPrototype}%
1427 \ExplSyntaxOn
1428 \seq_put_right:Nn
1429 \g_@@_renderers_seq
1430 { jekyllDataEnd }
1431 \prop_put:Nnn
1432 \g_@@_renderer_arities_prop
1433 { jekyllDataEnd }
1434 { 0 }
1435 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the mapping.

```

1436 \def\markdownRendererJekyllDataMappingBegin{%
1437 \markdownRendererJekyllDataMappingBeginPrototype}%

```

```

1438 \ExplSyntaxOn
1439 \seq_put_right:Nn
1440 \g_@@_renderers_seq
1441 { jekyllDataMappingBegin }
1442 \prop_put:Nnn
1443 \g_@@_renderer_arities_prop
1444 { jekyllDataMappingBegin }
1445 { 2 }
1446 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingEnd` macro represents the end of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1447 \def\markdownRendererJekyllDataMappingEnd{%
1448 \markdownRendererJekyllDataMappingEndPrototype}%
1449 \ExplSyntaxOn
1450 \seq_put_right:Nn
1451 \g_@@_renderers_seq
1452 { jekyllDataMappingEnd }
1453 \prop_put:Nnn
1454 \g_@@_renderer_arities_prop
1455 { jekyllDataMappingEnd }
1456 { 0 }
1457 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the sequence.

```

1458 \def\markdownRendererJekyllDataSequenceBegin{%
1459 \markdownRendererJekyllDataSequenceBeginPrototype}%
1460 \ExplSyntaxOn
1461 \seq_put_right:Nn
1462 \g_@@_renderers_seq
1463 { jekyllDataSequenceBegin }
1464 \prop_put:Nnn
1465 \g_@@_renderer_arities_prop
1466 { jekyllDataSequenceBegin }
1467 { 2 }
1468 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceEnd` macro represents the end of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1469 \def\markdownRendererJekyllDataSequenceEnd{%

```

```

1470 \markdownRendererJekyllDataSequenceEndPrototype}%
1471 \ExplSyntaxOn
1472 \seq_put_right:Nn
1473 \g_@@_renderers_seq
1474 { jekyllDataSequenceEnd }
1475 \prop_put:Nnn
1476 \g_@@_renderer_arities_prop
1477 { jekyllDataSequenceEnd }
1478 { 0 }
1479 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataBoolean` macro represents a boolean scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```

1480 \def\markdownRendererJekyllDataBoolean{%
1481 \markdownRendererJekyllDataBooleanPrototype}%
1482 \ExplSyntaxOn
1483 \seq_put_right:Nn
1484 \g_@@_renderers_seq
1485 { jekyllDataBoolean }
1486 \prop_put:Nnn
1487 \g_@@_renderer_arities_prop
1488 { jekyllDataBoolean }
1489 { 2 }
1490 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataNumber` macro represents a numeric scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```

1491 \def\markdownRendererJekyllDataNumber{%
1492 \markdownRendererJekyllDataNumberPrototype}%
1493 \ExplSyntaxOn
1494 \seq_put_right:Nn
1495 \g_@@_renderers_seq
1496 { jekyllDataNumber }
1497 \prop_put:Nnn
1498 \g_@@_renderer_arities_prop
1499 { jekyllDataNumber }
1500 { 2 }
1501 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option

is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```

1502 \def\markdownRendererJekyllDataString{%
1503 \markdownRendererJekyllDataStringPrototype}%
1504 \ExplSyntaxOn
1505 \seq_put_right:Nn
1506 \g_@@_renderers_seq
1507 { jekyllDataString }
1508 \prop_put:Nnn
1509 \g_@@_renderer_arities_prop
1510 { jekyllDataString }
1511 { 2 }
1512 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.4.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```

1513 \def\markdownRendererJekyllDataEmpty{%
1514 \markdownRendererJekyllDataEmptyPrototype}%
1515 \ExplSyntaxOn
1516 \seq_put_right:Nn
1517 \g_@@_renderers_seq
1518 { jekyllDataEmpty }
1519 \prop_put:Nnn
1520 \g_@@_renderer_arities_prop
1521 { jekyllDataEmpty }
1522 { 1 }
1523 \ExplSyntaxOff

```

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

```

1524 \def\markdownRendererHeadingOne{%
1525 \markdownRendererHeadingOnePrototype}%
1526 \ExplSyntaxOn
1527 \seq_put_right:Nn
1528 \g_@@_renderers_seq
1529 { headingOne }
1530 \prop_put:Nnn
1531 \g_@@_renderer_arities_prop
1532 { headingOne }
1533 { 1 }

```

1534 \ExplSyntaxOff

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

```
1535 \def\markdownRendererHeadingTwo{%
1536 \markdownRendererHeadingTwoPrototype}%
1537 \ExplSyntaxOn
1538 \seq_put_right:Nn
1539 \g_@@_renderers_seq
1540 { headingTwo }
1541 \prop_put:Nnn
1542 \g_@@_renderer_arities_prop
1543 { headingTwo }
1544 { 1 }
1545 \ExplSyntaxOff
```

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

```
1546 \def\markdownRendererHeadingThree{%
1547 \markdownRendererHeadingThreePrototype}%
1548 \ExplSyntaxOn
1549 \seq_put_right:Nn
1550 \g_@@_renderers_seq
1551 { headingThree }
1552 \prop_put:Nnn
1553 \g_@@_renderer_arities_prop
1554 { headingThree }
1555 { 1 }
1556 \ExplSyntaxOff
```

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

```
1557 \def\markdownRendererHeadingFour{%
1558 \markdownRendererHeadingFourPrototype}%
1559 \ExplSyntaxOn
1560 \seq_put_right:Nn
1561 \g_@@_renderers_seq
1562 { headingFour }
1563 \prop_put:Nnn
1564 \g_@@_renderer_arities_prop
1565 { headingFour }
1566 { 1 }
1567 \ExplSyntaxOff
```

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

```

1568 \def\markdownRendererHeadingFive{%
1569 \markdownRendererHeadingFivePrototype}%
1570 \ExplSyntaxOn
1571 \seq_put_right:Nn
1572 \g_@@_renderers_seq
1573 { headingFive }
1574 \prop_put:Nnn
1575 \g_@@_renderer_arities_prop
1576 { headingFive }
1577 { 1 }
1578 \ExplSyntaxOff

```

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

```

1579 \def\markdownRendererHeadingSix{%
1580 \markdownRendererHeadingSixPrototype}%
1581 \ExplSyntaxOn
1582 \seq_put_right:Nn
1583 \g_@@_renderers_seq
1584 { headingSix }
1585 \prop_put:Nnn
1586 \g_@@_renderer_arities_prop
1587 { headingSix }
1588 { 1 }
1589 \ExplSyntaxOff

```

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

```

1590 \def\markdownRendererHorizontalRule{%
1591 \markdownRendererHorizontalRulePrototype}%
1592 \ExplSyntaxOn
1593 \seq_put_right:Nn
1594 \g_@@_renderers_seq
1595 { horizontalRule }
1596 \prop_put:Nnn
1597 \g_@@_renderer_arities_prop
1598 { horizontalRule }
1599 { 0 }
1600 \ExplSyntaxOff

```

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

```

1601 \def\markdownRendererFootnote{%
1602 \markdownRendererFootnotePrototype}%

```

```

1603 \ExplSyntaxOn
1604 \seq_put_right:Nn
1605 \g_@@_renderers_seq
1606 { footnote }
1607 \prop_put:Nnn
1608 \g_@@_renderer_arities_prop
1609 { footnote }
1610 { 1 }
1611 \ExplSyntaxOff

```

**2.2.3.22 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 enabled. 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.

```

1612 \def\markdownRendererCite{%
1613 \markdownRendererCitePrototype}%
1614 \ExplSyntaxOn
1615 \seq_put_right:Nn
1616 \g_@@_renderers_seq
1617 { cite }
1618 \prop_put:Nnn
1619 \g_@@_renderer_arities_prop
1620 { cite }
1621 { 1 }
1622 \ExplSyntaxOff

```

**2.2.3.23 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 enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```

1623 \def\markdownRendererTextCite{%
1624 \markdownRendererTextCitePrototype}%
1625 \ExplSyntaxOn
1626 \seq_put_right:Nn
1627 \g_@@_renderers_seq
1628 { textCite }
1629 \prop_put:Nnn
1630 \g_@@_renderer_arities_prop
1631 { textCite }
1632 { 1 }
1633 \ExplSyntaxOff

```



**2.2.3.24 Table Renderer** The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters `{<caption>}{<number of rows>}{<number of columns>}` followed by `{<alignments>}` and then by `{<row>}` repeated `<number of rows>` times, where `<row>` is `{<column>}` repeated `<number of columns>` times, `<alignments>` is `<alignment>` repeated `<number of columns>` times, and `<alignment>` is one of the following:

- **d** – The corresponding column has an unspecified (default) alignment.
- **l** – The corresponding column is left-aligned.
- **c** – The corresponding column is centered.
- **r** – The corresponding column is right-aligned.

```

1634 \def\markdownRendererTable{%
1635 \markdownRendererTablePrototype}%
1636 \ExplSyntaxOn
1637 \seq_put_right:Nn
1638 \g_@@_renderers_seq
1639 { table }
1640 \prop_put:Nnn
1641 \g_@@_renderer_arities_prop
1642 { table }
1643 { 3 }
1644 \ExplSyntaxOff

```

**2.2.3.25 HTML Comment Renderers** The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

The `\markdownRendererBlockHtmlCommentBegin` and `\markdownRendererBlockHtmlCommentEnd` macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```

1645 \def\markdownRendererInlineHtmlComment{%
1646 \markdownRendererInlineHtmlCommentPrototype}%
1647 \ExplSyntaxOn
1648 \seq_put_right:Nn
1649 \g_@@_renderers_seq
1650 { inlineHtmlComment }
1651 \prop_put:Nnn
1652 \g_@@_renderer_arities_prop
1653 { inlineHtmlComment }
1654 { 1 }
1655 \ExplSyntaxOff
1656 \def\markdownRendererBlockHtmlCommentBegin{%
1657 \markdownRendererBlockHtmlCommentBeginPrototype}%
1658 \ExplSyntaxOn

```

```

1659 \seq_put_right:Nn
1660 \g_@@_renderers_seq
1661 { blockHtmlCommentBegin }
1662 \prop_put:Nnn
1663 \g_@@_renderer_arities_prop
1664 { blockHtmlCommentBegin }
1665 { 0 }
1666 \ExplSyntaxOff
1667 \def\markdownRendererBlockHtmlCommentEnd{%
1668 \markdownRendererBlockHtmlCommentEndPrototype}%
1669 \ExplSyntaxOn
1670 \seq_put_right:Nn
1671 \g_@@_renderers_seq
1672 { blockHtmlCommentEnd }
1673 \prop_put:Nnn
1674 \g_@@_renderer_arities_prop
1675 { blockHtmlCommentEnd }
1676 { 0 }
1677 \ExplSyntaxOff

```

**2.2.3.26 HTML Tag and Element Renderers** The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.

```

1678 \def\markdownRendererInlineHtmlTag{%
1679 \markdownRendererInlineHtmlTagPrototype}%
1680 \ExplSyntaxOn
1681 \seq_put_right:Nn
1682 \g_@@_renderers_seq
1683 { inlineHtmlTag }
1684 \prop_put:Nnn
1685 \g_@@_renderer_arities_prop
1686 { inlineHtmlTag }
1687 { 1 }
1688 \ExplSyntaxOff
1689 \def\markdownRendererInputBlockHtmlElement{%
1690 \markdownRendererInputBlockHtmlElementPrototype}%
1691 \ExplSyntaxOn
1692 \seq_put_right:Nn
1693 \g_@@_renderers_seq
1694 { inputBlockHtmlElement }

```

```

1695 \prop_put:Nnn
1696 \g_@@_renderer_arities_prop
1697 { inputBlockHtmlElement }
1698 { 1 }
1699 \ExplSyntaxOff

```

**2.2.3.27 Attribute Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

`\markdownRendererAttributeIdentifier` represents the  $\langle identifier \rangle$  of a markdown element (`id=" $\langle identifier \rangle$ "` in HTML and `# $\langle identifier \rangle$`  in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the  $\langle identifier \rangle$ .

`\markdownRendererAttributeClassName` represents the  $\langle class name \rangle$  of a markdown element (`class=" $\langle class name \rangle$  ..."` in HTML and `.. $\langle class name \rangle$`  in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

`\markdownRendererAttributeKeyValue` represents a HTML attribute in the form  $\langle key \rangle = \langle value \rangle$  that is neither an identifier nor a class name. The macro receives two attributes that correspond to the  $\langle key \rangle$  and the  $\langle value \rangle$ , respectively.

```

1700 \def\markdownRendererAttributeIdentifier{%
1701 \markdownRendererAttributeIdentifierPrototype}%
1702 \ExplSyntaxOn
1703 \seq_put_right:Nn
1704 \g_@@_renderers_seq
1705 { attributeIdentifier }
1706 \prop_put:Nnn
1707 \g_@@_renderer_arities_prop
1708 { attributeIdentifier }
1709 { 1 }
1710 \ExplSyntaxOff
1711 \def\markdownRendererAttributeClassName{%
1712 \markdownRendererAttributeClassNamePrototype}%
1713 \ExplSyntaxOn
1714 \seq_put_right:Nn
1715 \g_@@_renderers_seq
1716 { attributeClassName }
1717 \prop_put:Nnn
1718 \g_@@_renderer_arities_prop
1719 { attributeClassName }
1720 { 1 }
1721 \ExplSyntaxOff
1722 \def\markdownRendererAttributeKeyValue{%
1723 \markdownRendererAttributeKeyValuePrototype}%
1724 \ExplSyntaxOn
1725 \seq_put_right:Nn

```

```

1726 \g_@@_renderers_seq
1727 { attributeKeyValue }
1728 \prop_put:Nnn
1729 \g_@@_renderer_arities_prop
1730 { attributeKeyValue }
1731 { 2 }
1732 \ExplSyntaxOff

```

**2.2.3.28 Header Attribute Context Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttributeContextEnd` macros represent the beginning and the end of a section in which the attributes of a heading apply. The macros receive no arguments.

```

1733 \def\markdownRendererHeaderAttributeContextBegin{%
1734 \markdownRendererHeaderAttributeContextBeginPrototype}%
1735 \ExplSyntaxOn
1736 \seq_put_right:Nn
1737 \g_@@_renderers_seq
1738 { headerAttributeContextBegin }
1739 \prop_put:Nnn
1740 \g_@@_renderer_arities_prop
1741 { headerAttributeContextBegin }
1742 { 0 }
1743 \ExplSyntaxOff
1744 \def\markdownRendererHeaderAttributeContextEnd{%
1745 \markdownRendererHeaderAttributeContextEndPrototype}%
1746 \ExplSyntaxOn
1747 \seq_put_right:Nn
1748 \g_@@_renderers_seq
1749 { headerAttributeContextEnd }
1750 \prop_put:Nnn
1751 \g_@@_renderer_arities_prop
1752 { headerAttributeContextEnd }
1753 { 0 }
1754 \ExplSyntaxOff

```

**2.2.3.29 Strike-Through Renderer** The `\markdownRendererStrikeThrough` macro represents a strike-through span of text. The macro receives a single argument that corresponds to the striked-out span of text. This macro will only be produced, when the `strikeThrough` option is enabled.

```

1755 \def\markdownRendererStrikeThrough{%
1756 \markdownRendererStrikeThroughPrototype}%
1757 \ExplSyntaxOn
1758 \seq_put_right:Nn

```

```

1759 \g_@@_renderers_seq
1760 { strikeThrough }
1761 \prop_put:Nnn
1762 \g_@@_renderer_arities_prop
1763 { strikeThrough }
1764 { 1 }
1765 \ExplSyntaxOff

```

**2.2.3.30 Superscript Renderer** The `\markdownRendererSuperscript` macro represents a superscript span of text. The macro receives a single argument that corresponds to the superscript span of text. This macro will only be produced, when the `superscripts` option is enabled.

```

1766 \def\markdownRendererSuperscript{%
1767 \markdownRendererSuperscriptPrototype}%
1768 \ExplSyntaxOn
1769 \seq_put_right:Nn
1770 \g_@@_renderers_seq
1771 { superscript }
1772 \prop_put:Nnn
1773 \g_@@_renderer_arities_prop
1774 { superscript }
1775 { 1 }
1776 \ExplSyntaxOff

```

**2.2.3.31 Subscript Renderer** The `\markdownRendererSubscript` macro represents a subscript span of text. The macro receives a single argument that corresponds to the subscript span of text. This macro will only be produced, when the `subscripts` option is enabled.

```

1777 \def\markdownRendererSubscript{%
1778 \markdownRendererSubscriptPrototype}%
1779 \ExplSyntaxOn
1780 \seq_put_right:Nn
1781 \g_@@_renderers_seq
1782 { subscript }
1783 \prop_put:Nnn
1784 \g_@@_renderer_arities_prop
1785 { subscript }
1786 { 1 }
1787 \ExplSyntaxOff

```

## 2.2.4 Token Renderer Prototypes

**2.2.4.1 YAML Metadata Renderer Prototypes** By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the  $\text{\LaTeX}$ 3 kernel.

```
1788 \ExplSyntaxOn
1789 \keys_define:nn
1790 { markdown/jekyllData }
1791 { }
1792 \ExplSyntaxOff
```

The following  $\text{\TeX}$  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  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  implementations (see sections 3.3 and 3.4).

```
1793 \ExplSyntaxOn
1794 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
1795 {
1796 \seq_map_function:NN
1797 \g_@@_renderers_seq
1798 \@@_plaintex_define_renderer_prototype:n
1799 \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
1800 \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1801 \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1802 \let\markdownRendererBlockHtmlCommentEnd=\fi
1803 }
1804 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
1805 {
1806 \@@_renderer_prototype_tl_to_csname:nN
1807 { #1 }
1808 \l_tmpa_tl
1809 \prop_get:NnN
1810 \g_@@_renderer_arities_prop
1811 { #1 }
1812 \l_tmpb_tl
1813 \@@_plaintex_define_renderer_prototype:cV
1814 { \l_tmpa_tl }
1815 \l_tmpb_tl
1816 }
1817 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
1818 {
1819 \tl_set:Nn
1820 \l_tmpa_tl
1821 { \str_uppercase:n { #1 } }
1822 \tl_set:Nx
1823 #2
```

```

1824 {
1825 markdownRenderer
1826 \tl_head:f { \l_tmpa_tl }
1827 \tl_tail:n { #1 }
1828 Prototype
1829 }
1830 }
1831 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
1832 {
1833 \cs_generate_from_arg_count:NNnn
1834 #1
1835 \cs_set:Npn
1836 { #2 }
1837 { }
1838 }
1839 \cs_generate_variant:Nn
1840 \@@_plaintex_define_renderer_prototype:Nn
1841 { cV }
1842 \@@_plaintex_define_renderer_prototypes:
1843 \ExplSyntaxOff

```

### 2.2.5 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. 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 T<sub>E</sub>X engine that does not support direct Lua access is starting to buffer a text. The plain T<sub>E</sub>X implementation changes the category code of plain T<sub>E</sub>X 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*.

```

1844 \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 T<sub>E</sub>X 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.

```

1845 \let\markdownReadAndConvert\relax

```

```
1846 \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*.

```
1847 \catcode`\|=0\catcode`\=12%
1848 |gdef|markdownBegin{%
1849 |markdownReadAndConvert{\markdownEnd}%
1850 {|markdownEnd}}%
1851 |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
- `3` – The `lt3luabridge` Lua package

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.

The `\markdownMode` macro has been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
1852 \ExplSyntaxOn
1853 \cs_if_exist:NF
1854 \markdownMode
1855 {
1856 \file_if_exist:nTF
1857 { lt3luabridge.tex }
1858 {
1859 \cs_new:Npn
1860 \markdownMode
1861 { 3 }
1862 }
1863 {
1864 \cs_if_exist:NTF
1865 \directlua
1866 {
1867 \cs_new:Npn
1868 \markdownMode
1869 { 2 }

```



```

1870 }
1871 {
1872 \cs_new:Npn
1873 \markdownMode
1874 { 0 }
1875 }
1876 }
1877 }

```

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros have been deprecated and will be removed in Markdown 3.0.0:

```

1878 \def\markdownLuaRegisterIBCallback#1{\relax}%
1879 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

1880 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
1881 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
1882 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
1883 \input markdown/markdown

```

The L<sup>A</sup>T<sub>E</sub>X interface is implemented by the `markdown.sty` file, which can be loaded from the L<sup>A</sup>T<sub>E</sub>X document preamble as follows:

```
\usepackage[<options>]{markdown}
```

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts

L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
1884 \newenvironment{markdown}\relax\relax
1885 \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` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `markdown` and `markdown*` environments:

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

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 T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `\markdownInput` macro:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[smartEllipses]{hello.md}
\end{document}
```

### 2.3.2 Options

The L<sup>A</sup>T<sub>E</sub>X options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted.

Except for the `plain` option described in Section 2.3.2.1, and the L<sup>A</sup>T<sub>E</sub>X themes described in Section 2.3.2.2, and the L<sup>A</sup>T<sub>E</sub>X setup snippets described in Section 2.3.2.3, L<sup>A</sup>T<sub>E</sub>X options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain T<sub>E</sub>X interface (see Sections 2.2.3 and 2.2.4).

The L<sup>A</sup>T<sub>E</sub>X options may be specified when loading the L<sup>A</sup>T<sub>E</sub>X package, when using the `markdown*` L<sup>A</sup>T<sub>E</sub>X environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```

1886 \ExplSyntaxOn
1887 \cs_new:Nn
1888 \@@_setup:n
1889 {
1890 \keys_set:nn
1891 { markdown/latex-options }
1892 { #1 }
1893 }
1894 \let\markdownSetup=\@@_setup:n
1895 \ExplSyntaxOff

```

We may also store L<sup>A</sup>T<sub>E</sub>X options as *setup snippets* and invoke them later using the `\markdownSetupSnippet` macro. The `\markdownSetupSnippet` macro receives two arguments: the name of the setup snippet and the options to store:

```

1896 \newcommand\markdownSetupSnippet[2]{%
1897 \markdownIfSnippetExists{#1}%
1898 {%
1899 \markdownWarning
1900 {Redefined setup snippet \markdownLaTeXThemeName#1}%
1901 \csname markdownLaTeXSetupSnippet%
1902 \markdownLaTeXThemeName#1\endcsname={#2}%
1903 }{%
1904 \newtoks\next
1905 \next={#2}%
1906 \expandafter\let\csname markdownLaTeXSetupSnippet%
1907 \markdownLaTeXThemeName#1\endcsname=\next
1908 }}%

```

To decide whether a setup snippet exists, we can use the `\markdownIfSnippetExists` macro:

```

1909 \newcommand\markdownIfSnippetExists[3]{%
1910 \ifundefined
1911 {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
1912 {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and L<sup>A</sup>T<sub>E</sub>X themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of L<sup>A</sup>T<sub>E</sub>X options, we will maintain the `\g_@@_latex_options_seq` sequence.

```
1913 \ExplSyntaxOn
```

```
1914 \seq_new:N \g_@@_latex_options_seq
```

To enable the reflection of default L<sup>A</sup>T<sub>E</sub>X options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```
1915 \prop_new:N \g_@@_latex_option_types_prop
```

```
1916 \prop_new:N \g_@@_default_latex_options_prop
```

```
1917 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
```

```
1918 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
```

```
1919 \cs_new:Nn
```

```
1920 \@@_add_latex_option:nnn
```

```
1921 {
```

```
1922 \@@_add_option:Vnnn
```

```
1923 \c_@@_option_layer_latex_tl
```

```
1924 { #1 }
```

```
1925 { #2 }
```

```
1926 { #3 }
```

```
1927 }
```

**2.3.2.1 No default token renderer prototypes** Default token renderer prototypes require L<sup>A</sup>T<sub>E</sub>X packages that may clash with other packages used in a document. Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain T<sub>E</sub>X implementation (see Section 3.2.2) and prevent the soft L<sup>A</sup>T<sub>E</sub>X prerequisites in Section 1.1.3 from being loaded: The plain option must be set before or when loading the package. Setting the option after loading the package will have no effect.

```
\usepackage[plain]{markdown}
```

```
1928 \@@_add_latex_option:nnn
```

```
1929 { plain }
```

```
1930 { boolean }
```

```
1931 { false }
```

```
1932 \ExplSyntaxOff
```

**2.3.2.2 L<sup>A</sup>T<sub>E</sub>X themes** User-defined L<sup>A</sup>T<sub>E</sub>X themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L<sup>A</sup>T<sub>E</sub>X packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The  $\text{\LaTeX}$  option `theme=<theme name>` loads a  $\text{\LaTeX}$  package (further referred to as a *theme*) named `markdowntheme<munged theme name>.sty`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_), the *theme name* is *qualified* and contains no underscores, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [6, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single  $\text{\LaTeX}$  document class or for a single  $\text{\LaTeX}$  package. The preferred format of a theme name is `<theme author>/<target  $\text{\LaTeX}$  document class or package>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged, because  $\text{\LaTeX}$  packages are identified only by their filenames, not by their pathnames. [7] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a  $\text{\LaTeX}$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\text{\LaTeX}$  option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown  $\text{\LaTeX}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty`  $\text{\LaTeX}$  package, and finally the `markdownthemewitiko_dot.sty`  $\text{\LaTeX}$  package:

```
\usepackage[
 theme = witiko/beamer/MU,
 theme = witiko/dot,
]{markdown}
```

```
1933 \newif\ifmarkdownLaTeXLoaded
1934 \markdownLaTeXLoadedfalse
1935 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
1936 \ExplSyntaxOn
1937 \tl_new:N \markdownLaTeXThemePackageName
1938 \cs_new:Nn
1939 \@@_set_latex_theme:n
1940 {
1941 \str_if_in:nnF
1942 { #1 }
```

```

1943 { / }
1944 {
1945 \markdownError
1946 { Won't~load~theme~with~unqualified~name~#1 }
1947 { Theme~names~must~contain~at~least~one~forward~slash }
1948 }
1949 \str_if_in:nnT
1950 { #1 }
1951 { _ }
1952 {
1953 \markdownError
1954 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1955 { Theme~names~must~not~contain~underscores~in~their~names }
1956 }
1957 \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
1958 \str_replace_all:Nnn
1959 \markdownLaTeXThemePackageName
1960 { / }
1961 { _ }
1962 \edef\markdownLaTeXThemePackageName{
1963 markdowntheme\markdownLaTeXThemePackageName}
1964 \expandafter\markdownLaTeXThemeLoad\expandafter{
1965 \markdownLaTeXThemePackageName}{#1/}
1966 }
1967 \keys_define:nn
1968 { markdown/latex-options }
1969 {
1970 theme .code:n = { \@@_set_latex_theme:n { #1 } },
1971 }
1972 \ExplSyntaxOff

```

The L<sup>A</sup>T<sub>E</sub>X themes have a useful synergy with the setup snippets (see Section 2.3.2): To make it less likely that different themes will define setup snippets with the same name, we will prepend *<theme name>/* before the snippet name and use the result as the snippet name. For example, if the **witiko/dot** theme defines the **product** setup snippet, the setup snippet will be available under the name **witiko/dot/product**. Due to limitations of L<sup>A</sup>T<sub>E</sub>X, themes may not be loaded after the beginning of a L<sup>A</sup>T<sub>E</sub>X document.

```

1973 \ExplSyntaxOn
1974 \@onlypreamble
1975 \@@_set_latex_theme:n
1976 \ExplSyntaxOff

```

Example themes provided with the Markdown package include:

**witiko/dot** A theme that typesets fenced code blocks with the **dot ...** infostring as

images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

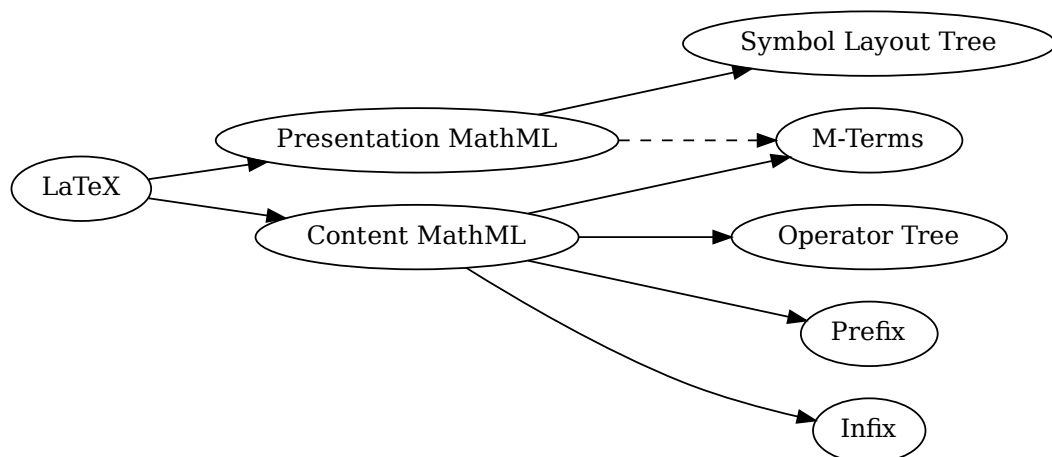
```
\documentclass{article}
\usepackage[theme=witiko/dot]{markdown}
\setkeys{Gin}{
 width = \columnwidth,
 height = 0.65\paperheight,
 keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

  latex -> pmml;
  latex -> cmml;
  pmml -> slt;
  cmml -> opt;
  cmml -> prefix;
  cmml -> infix;
  pmml -> mterms [style=dashed];
  cmml -> mterms;

  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
```
\end{markdown}
\end{document}
```

Typesetting the above document produces the output shown in Figure 4.

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain  $\text{\TeX}$  option is enabled.



**Figure 4: Various formats of mathematical formulae**

1977 \ProvidesPackage{markdownthemewitiko\_dot}[2021/03/09]%

**witiko/graphicx/http** A theme that adds support for downloading images whose URL has the http or https protocol.

```

\documentclass{article}
\usepackage[theme=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}
! [img] (https://github.com/witiko/markdown/raw/main/markdown.png
 "The banner of the Markdown package")
\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils **md5sum** and either GNU Wget or cURL installed. The theme also requires shell access unless the **\markdownOptionFrozenCache** plain  $\TeX$  option is enabled.

1978 \ProvidesPackage{markdownthemewitiko\_graphicx\_http}[2021/03/22]%

**witiko/tilde** A theme that makes tilde (~) always typeset the non-breaking space even when the **hybrid** Lua option is disabled.



```

\documentclass{book}
\usepackage{markdown}
\markdownSetup{pipeTables,tableCaptions}
\begin{document}
\begin{markdown}
Introduction
=====
Section
Subsection
Hello *Markdown*!

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

: Table
\end{markdown}
\end{document}

```



# Chapter 1

## Introduction

1.1 Section

1.1.1 Subsection

Hello *Markdown*!

| Right | Left | Default | Center |
|-------|------|---------|--------|
| 12    | 12   | 12      | 12     |
| 123   | 123  | 123     | 123    |
| 1     | 1    | 1       | 1      |

Table 1.1: Table

Figure 5: The banner of the Markdown package

```

\documentclass{article}
\usepackage[theme=witiko/tilde]{markdown}
\begin{document}
\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}

```

Typesetting the above document produces the following text: “Bartel Leendert van der Waerden”.

1979 \ProvidesPackage{markdownthemewitiko\_tilde}[2021/03/22]%

Please, see Section 3.3.2.1 for implementation details of the example themes.

**2.3.2.3 L<sup>A</sup>T<sub>E</sub>X setup snippets** The L<sup>A</sup>T<sub>E</sub>X option with key **snippet** invokes a snippet named *<value>*:

```

1980 \ExplSyntaxOn
1981 \keys_define:nn
1982 { markdown/latex-options }

```

```

1983 {
1984 snippet .code:n = {
1985 \markdownIfSnippetExists{#1}
1986 {
1987 \expandafter\markdownSetup\expandafter{
1988 \the\csname markdownLaTeXSetupSnippet
1989 \markdownLaTeXThemeName#1\endcsname}
1990 }{
1991 \markdownError
1992 {Can't~invoke~setup~snippet~#1}
1993 {The~setup~snippet~is~undefined}
1994 }
1995 }
1996 }
1997 \ExplSyntaxOff

```

Here is how we can use setup snippets to store options and invoke them later:

```

\markdownSetupSnippet{romanNumerals}{
 renderers = {
 olItemWithNumber = {\item[\romannumeral#1\relax.]},
 },
}
\begin{markdown}

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

\end{markdown}
\begin{markdown*}{snippet=romanNumerals}

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown*}

```

**2.3.2.4 Plain T<sub>E</sub>X Interface Options** Here, we automatically define plain T<sub>E</sub>X macros and the  $\langle key \rangle = \langle value \rangle$  interface for the above L<sup>A</sup>T<sub>E</sub>X options.

```

1998 \ExplSyntaxOn
1999 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:

```

```

2000 {
2001 \seq_map_inline:Nn
2002 \g_@@_latex_options_seq
2003 {
2004 \@@_plain_tex_define_option_command:n
2005 { ##1 }
2006 }

```

Furthermore, we also define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```

2007 \seq_map_inline:Nn
2008 \g_@@_option_layers_seq
2009 {
2010 \seq_map_inline:cn
2011 { g_@@_ ##1 _options_seq }
2012 {
2013 \@@_latex_define_option_keyval:nn
2014 { ##1 }
2015 { #####1 }
2016 }
2017 }
2018 }
2019 \cs_new:Nn \@@_latex_define_option_keyval:nn
2020 {
2021 \prop_get:cnN
2022 { g_@@_ #1 _option_types_prop }
2023 { #2 }
2024 \l_tmpa_tl
2025 \keys_define:nn
2026 { markdown/latex-options }
2027 {
2028 #2 .code:n = {
2029 \@@_set_option_value:nn
2030 { #2 }
2031 { ##1 }
2032 },
2033 }
2034 \str_if_eq:VVT
2035 \l_tmpa_tl
2036 \c_@@_option_type_boolean_tl
2037 {
2038 \keys_define:nn
2039 { markdown/latex-options }
2040 {
2041 #2 .default:n = { true },
2042 }
2043 }

```

For options of type `clist`, we assume that  $\langle key \rangle$  is a regular English noun in plural (such as `extensions`) and we also define the  $\langle singular\ key \rangle = \langle value \rangle$  interface, where  $\langle singular\ key \rangle$  is  $\langle key \rangle$  after stripping the trailing -s (such as `extension`). Rather than setting the option to  $\langle value \rangle$ , this interface appends  $\langle value \rangle$  to the current value as the rightmost item in the list.

```

2044 \str_if_eq:VVT
2045 \l_tmpa_tl
2046 \c_@@_option_type_clist_tl
2047 {
2048 \tl_set:Nn
2049 \l_tmpa_tl
2050 { #2 }
2051 \tl_reverse:N
2052 \l_tmpa_tl
2053 \str_if_eq:enF
2054 {
2055 \tl_head:V
2056 \l_tmpa_tl
2057 }
2058 { s }
2059 {
2060 \msg_error:nnn
2061 { @@ }
2062 { malformed-name-for-clist-option }
2063 { #2 }
2064 }
2065 \tl_set:Nx
2066 \l_tmpa_tl
2067 {
2068 \tl_tail:V
2069 \l_tmpa_tl
2070 }
2071 \tl_reverse:N
2072 \l_tmpa_tl
2073 \tl_put_right:Nn
2074 \l_tmpa_tl
2075 {
2076 .code:n = {
2077 \@@_get_option_value:nN
2078 { #2 }
2079 \l_tmpa_tl
2080 \clist_set:NV
2081 \l_tmpa_clist
2082 { \l_tmpa_tl, { ##1 } }
2083 \@@_set_option_value:nV
2084 { #2 }

```

```

2085 \l_tmpa_clist
2086 }
2087 }
2088 \keys_define:nV
2089 { markdown/latex-options }
2090 \l_tmpa_tl
2091 }
2092 }
2093 \cs_generate_variant:Nn
2094 \clist_set:Nn
2095 { NV }
2096 \cs_generate_variant:Nn
2097 \keys_define:nn
2098 { nV }
2099 \cs_generate_variant:Nn
2100 \@@_set_option_value:nn
2101 { nV }
2102 \prg_generate_conditional_variant:Nnn
2103 \str_if_eq:nn
2104 { en }
2105 { F }
2106 \msg_new:nnn
2107 { @@ }
2108 { malformed-name-for-clist-option }
2109 {
2110 Clist~option~name~#1~does~not~end~with~-s.
2111 }
2112 \@@_latex_define_option_commands_and_keyvals:
2113 \ExplSyntaxOff

```

The `\markdownOptionFinalizeCache` and `\markdownOptionFrozenCache` plain TeX options are exposed through L<sup>A</sup>T<sub>E</sub>X options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [8, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```

\documentclass[frozenscache]{article}
\usepackage{markdown,minted}
\begin{document}
\end{document}

```

We hope that other packages will support the `finalizcache` and `frozenscache`

package options in the future, so that they can become a standard interface for preparing L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```
2114 \DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
2115 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of plain T<sub>E</sub>X interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

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

**2.3.2.5 Plain T<sub>E</sub>X Markdown Token Renderers** The L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X interface (see Section 2.2.3).

```
2116 \ExplSyntaxOn
2117 \cs_new:Nn \@@_latex_define_renderers:
2118 {
2119 \seq_map_function:NN
2120 \g_@@_renderers_seq
2121 \@@_latex_define_renderer:n
2122 }
2123 \cs_new:Nn \@@_latex_define_renderer:n
2124 {
2125 \@@_renderer_tl_to_csname:nN
2126 { #1 }
2127 \l_tmpa_tl
2128 \prop_get:NnN
2129 \g_@@_renderer_arities_prop
2130 { #1 }
2131 \l_tmpb_tl
2132 \@@_latex_define_renderer:ncV
2133 { #1 }
2134 { \l_tmpa_tl }
2135 \l_tmpb_tl
2136 }
2137 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2138 {
2139 \tl_set:Nn
2140 \l_tmpa_tl
2141 { \str_uppercase:n { #1 } }
```

```

2142 \tl_set:Nx
2143 #2
2144 {
2145 markdownRenderer
2146 \tl_head:f { \l_tmpa_tl }
2147 \tl_tail:n { #1 }
2148 }
2149 }
2150 \cs_new:Nn \@@_latex_define_renderer:nNn
2151 {
2152 \keys_define:nn
2153 { markdown/latex-options/renderers }
2154 {
2155 #1 .code:n = {
2156 \cs_generate_from_arg_count:NNnn
2157 #2
2158 \cs_set:Npn
2159 { #3 }
2160 { ##1 }
2161 },
2162 }
2163 }
2164 \cs_generate_variant:Nn
2165 \@@_latex_define_renderer:nNn
2166 { ncV }
2167 \ExplSyntaxOff

```

The following example  $\text{\LaTeX}$  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.
 }
}

```

**2.3.2.6 Plain  $\text{\TeX}$  Markdown Token Renderer Prototypes** The  $\text{\LaTeX}$  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  $\text{\TeX}$  interface (see Section 2.2.4).

```

2168 \ExplSyntaxOn
2169 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2170 {

```

```

2171 \seq_map_function:NN
2172 \g_@@_renderers_seq
2173 \@@_latex_define_renderer_prototype:n
2174 }
2175 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2176 {
2177 \@@_renderer_prototype_tl_to_csname:nN
2178 { #1 }
2179 \l_tmpa_tl
2180 \prop_get:NnN
2181 \g_@@_renderer_arities_prop
2182 { #1 }
2183 \l_tmpb_tl
2184 \@@_latex_define_renderer_prototype:ncV
2185 { #1 }
2186 { \l_tmpa_tl }
2187 \l_tmpb_tl
2188 }
2189 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2190 {
2191 \keys_define:nn
2192 { markdown/latex-options/renderer-prototypes }
2193 {
2194 #1 .code:n = {
2195 \cs_generate_from_arg_count:NNnn
2196 #2
2197 \cs_set:Npn
2198 { #3 }
2199 { ##1 }
2200 },
2201 }
2202 }
2203 \cs_generate_variant:Nn
2204 \@@_latex_define_renderer_prototype:nNn
2205 { ncV }
2206 \ExplSyntaxOff

```

The following example L<sup>A</sup>T<sub>E</sub>X 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 ConT<sub>E</sub>Xt Interface

The ConT<sub>E</sub>Xt interface provides a start-stop macro pair for the typesetting of markdown input from within ConT<sub>E</sub>Xt and facilities for setting Lua, plain T<sub>E</sub>X, and ConT<sub>E</sub>Xt options used during the conversion from markdown to plain T<sub>E</sub>X. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```
2207 \writestatus{loading}{ConTEXt User Module / markdown}%
2208 \startmodule[markdown]
2209 \unprotect
```

The ConT<sub>E</sub>Xt implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the ConT<sub>E</sub>Xt `\writestatus` macro.

```
2210 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2211 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2212 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2213 \do\#\do\~\do_ \do\% \do\~}%
2214 \input markdown/markdown
```

The ConT<sub>E</sub>Xt interface is implemented by the `t-markdown.tex` ConT<sub>E</sub>Xt module file that can be loaded as follows:

```
\usemodule[t] [markdown]
```

It is expected that the special plain T<sub>E</sub>X characters have the expected category codes, when `\input`ting the file.

### 2.4.1 Typesetting Markdown

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

```
2215 \let\startmarkdown\relax
2216 \let\stopmarkdown\relax
2217 \let\inputmarkdown\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 T<sub>E</sub>X interface.

The following example ConT<sub>E</sub>Xt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```

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

```

The `\inputmarkdown` 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  $\text{\TeX}$ . Unlike the `\markdownInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts Con $\text{\TeX}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{\LaTeX}$  code showcases the usage of the `\markdownInput` macro:

```

\usemodule[t] [markdown]
\starttext
\inputmarkdown[smartEllipses]{hello.md}
\stoptext

```

## 2.4.2 Options

The Con $\text{\TeX}$ t options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  (or, equivalently,  $\langle key \rangle = \text{yes}$ ) if the  $= \langle value \rangle$  part has been omitted.

Con $\text{\TeX}$ t options map directly to the options recognized by the plain  $\text{\TeX}$  interface (see Section 2.2.2).

The Con $\text{\TeX}$ t options may be specified when using the `\inputmarkdown` macro (see Section 2.4), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```

2218 \ExplSyntaxOn
2219 \cs_new:Nn
2220 \@@_setup:n
2221 {
2222 \keys_set:nn
2223 { markdown/context-options }
2224 { #1 }
2225 }
2226 \long\def\setupmarkdown[#1]
2227 {
2228 \@@_setup:n

```

```

2229 { #1 }
2230 }
2231 \ExplSyntaxOff

```

**2.4.2.1 ConT<sub>E</sub>Xt Interface Options** We define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```

2232 \ExplSyntaxOn
2233 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2234 {
2235 \seq_map_inline:Nn
2236 \g_@@_option_layers_seq
2237 {
2238 \seq_map_inline:cn
2239 { g_@@_ ##1 _options_seq }
2240 {
2241 \@@_context_define_option_keyval:nn
2242 { ##1 }
2243 { ####1 }
2244 }
2245 }
2246 }
2247 \cs_new:Nn \@@_context_define_option_keyval:nn
2248 {
2249 \prop_get:cnN
2250 { g_@@_ #1 _option_types_prop }
2251 { #2 }
2252 \l_tmpa_tl
2253 \keys_define:nn
2254 { markdown/context-options }
2255 {
2256 #2 .code:n = {
2257 \tl_set:Nx
2258 \l_tmpa_tl
2259 {
2260 \str_case:nnF
2261 { ##1 }
2262 {
2263 { yes } { true }
2264 { no } { false }
2265 }
2266 { ##1 }
2267 }
2268 \@@_set_option_value:nV
2269 { #2 }
2270 \l_tmpa_tl
2271 },

```

```

2272 }
2273 \str_if_eq:VVT
2274 \l_tmpa_tl
2275 \c_@@_option_type_boolean_tl
2276 {
2277 \keys_define:nn
2278 { markdown/context-options }
2279 {
2280 #2 .default:n = { true },
2281 }
2282 }
2283 }
2284 \cs_generate_variant:Nn
2285 \@@_set_option_value:nn
2286 { nV }
2287 \@@_context_define_option_commands_and_keyvals:
2288 \ExplSyntaxOff

```

## 3 Implementation

This part of the documentation 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.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{\TeX}$  *token renderers* is performed by the Lua layer. The plain  $\text{\TeX}$  layer provides default definitions for the token renderers. The  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  layers correct idiosyncrasies of the respective  $\text{\TeX}$  formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements **writer** and **reader** objects, which provide the conversion from markdown to plain  $\text{\TeX}$ , and **extensions** objects, which provide syntax extensions for the **writer** and **reader** objects.

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  $\text{\TeX}$  writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

2289 local upper, gsub, format, length =
2290 string.upper, string.gsub, string.format, string.len
2291 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2292 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2293 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  $\text{\TeX}$  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.

```
2294 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.

```
2295 function util.err(msg, exit_code)
2296 io.stderr:write("markdown.lua: " .. msg .. "\n")
2297 os.exit(exit_code or 1)
2298 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.

```
2299 function util.cache(dir, string, salt, transform, suffix)
2300 local digest = md5.sumhexa(string .. (salt or ""))
2301 local name = util.pathname(dir, digest .. suffix)
2302 local file = io.open(name, "r")
2303 if file == nil then -- If no cache entry exists, then create a new one.
2304 file = assert(io.open(name, "w"),
2305 [[Could not open file]] .. name .. [[for writing]])
2306 local result = string
2307 if transform ~= nil then
2308 result = transform(result)
2309 end
2310 assert(file:write(result))
2311 assert(file:close())
2312 end
2313 return name
2314 end
```

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

```
2315 function util.table_copy(t)
2316 local u = { }
2317 for k, v in pairs(t) do u[k] = v end
2318 return setmetatable(u, getmetatable(t))
2319 end
```

The `util.encode_json_string` method encodes a string `s` in JSON.

```
2320 function util.encode_json_string(s)
2321 s = s:gsub([[\\]], [[\\]])
2322 s = s:gsub([["]], [[\"]])
2323 return [[]] .. s .. [[]]
```

2324 end

The `util.lookup_files` method looks up files with filename `f` and returns its path. If the `kpathsea` library is available, it will search for files not only in the current working directory but also in the `TEX` directory structure. Further options for `kpathsea` can be specified in table `options`. [1, Section 10.7.4]

```
2325 util.lookup_files = (function()
2326 local ran_ok, kpse = pcall(require, "kpse")
2327 if ran_ok then
2328 kpse.set_program_name("luatex")
2329 else
2330 kpse = { lookup = function(f, _) return f end }
2331 end
2332
2333 local function lookup_files(f, options)
2334 return kpse.lookup(f, options)
2335 end
2336
2337 return lookup_files
2338 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 Ierusalimschy [9, Chapter 21].

```
2339 function util.expand_tabs_in_line(s, tabstop)
2340 local tab = tabstop or 4
2341 local corr = 0
2342 return (s:gsub("()\t", function(p)
2343 local sp = tab - (p - 1 + corr) % tab
2344 corr = corr - 1 + sp
2345 return string.rep(" ", sp)
2346 end))
2347 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.

```
2348 function util.walk(t, f)
2349 local typ = type(t)
2350 if typ == "string" then
2351 f(t)
2352 elseif typ == "table" then
2353 local i = 1
2354 local n
2355 n = t[i]
```

```

2356 while n do
2357 util.walk(n, f)
2358 i = i + 1
2359 n = t[i]
2360 end
2361 elseif typ == "function" then
2362 local ok, val = pcall(t)
2363 if ok then
2364 util.walk(val, f)
2365 end
2366 else
2367 f(tostring(t))
2368 end
2369 end

```

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

```

2370 function util.flatten(ary)
2371 local new = {}
2372 for _,v in ipairs(ary) do
2373 if type(v) == "table" then
2374 for _,w in ipairs(util.flatten(v)) do
2375 new[#new + 1] = w
2376 end
2377 else
2378 new[#new + 1] = v
2379 end
2380 end
2381 return new
2382 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.

```

2383 function util.rope_to_string(rope)
2384 local buffer = {}
2385 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2386 return table.concat(buffer)
2387 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.

```

2388 function util.rope_last(rope)
2389 if #rope == 0 then
2390 return nil
2391 else
2392 local l = rope[#rope]
2393 if type(l) == "table" then
2394 return util.rope_last(l)

```

```

2395 else
2396 return 1
2397 end
2398 end
2399 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  $1 \leq i \leq \#ary$ .

```

2400 function util.intersperse(ary, x)
2401 local new = {}
2402 local l = #ary
2403 for i,v in ipairs(ary) do
2404 local n = #new
2405 new[n + 1] = v
2406 if i ~= l then
2407 new[n + 2] = x
2408 end
2409 end
2410 return new
2411 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  $1 \leq i \leq \#ary$ .

```

2412 function util.map(ary, f)
2413 local new = {}
2414 for i,v in ipairs(ary) do
2415 new[i] = f(v)
2416 end
2417 return new
2418 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.

```

2419 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

2420 local char_escapes_list = ""
2421 for i,_ in pairs(char_escapes) do
2422 char_escapes_list = char_escapes_list .. i
2423 end

```

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

```

2424 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.

```
2425 if string_escapes then
2426 for k,v in pairs(string_escapes) do
2427 escapable = P(k) / v + escapable
2428 end
2429 end
```

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

```
2430 local escape_string = Cs((escapable + any)^0)
```

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

```
2431 return function(s)
2432 return lpeg.match(escape_string, s)
2433 end
2434 end
```

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

```
2435 function util.pathname(dir, file)
2436 if #dir == 0 then
2437 return file
2438 else
2439 return dir .. "/" .. file
2440 end
2441 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.

```
2442 local entities = {}
2443
2444 local character_entities = {
2445 ["Tab"] = 9,
2446 ["NewLine"] = 10,
2447 ["excl"] = 33,
```

```

2448 ["quot"] = 34,
2449 ["QUOT"] = 34,
2450 ["num"] = 35,
2451 ["dollar"] = 36,
2452 ["percent"] = 37,
2453 ["amp"] = 38,
2454 ["AMP"] = 38,
2455 ["apos"] = 39,
2456 ["lpar"] = 40,
2457 ["rpar"] = 41,
2458 ["ast"] = 42,
2459 ["midast"] = 42,
2460 ["plus"] = 43,
2461 ["comma"] = 44,
2462 ["period"] = 46,
2463 ["sol"] = 47,
2464 ["colon"] = 58,
2465 ["semi"] = 59,
2466 ["lt"] = 60,
2467 ["LT"] = 60,
2468 ["equals"] = 61,
2469 ["gt"] = 62,
2470 ["GT"] = 62,
2471 ["quest"] = 63,
2472 ["commat"] = 64,
2473 ["lsqb"] = 91,
2474 ["lbrack"] = 91,
2475 ["bsol"] = 92,
2476 ["rsqb"] = 93,
2477 ["rbrack"] = 93,
2478 ["Hat"] = 94,
2479 ["lowbar"] = 95,
2480 ["grave"] = 96,
2481 ["DiacriticalGrave"] = 96,
2482 ["lcub"] = 123,
2483 ["lbrace"] = 123,
2484 ["verbar"] = 124,
2485 ["vert"] = 124,
2486 ["VerticalLine"] = 124,
2487 ["rcub"] = 125,
2488 ["rbrace"] = 125,
2489 ["nbsp"] = 160,
2490 ["NonBreakingSpace"] = 160,
2491 ["iexcl"] = 161,
2492 ["cent"] = 162,
2493 ["pound"] = 163,
2494 ["curren"] = 164,

```

```

2495 ["yen"] = 165,
2496 ["brvbar"] = 166,
2497 ["sect"] = 167,
2498 ["Dot"] = 168,
2499 ["die"] = 168,
2500 ["DoubleDot"] = 168,
2501 ["uml"] = 168,
2502 ["copy"] = 169,
2503 ["COPY"] = 169,
2504 ["ordf"] = 170,
2505 ["laquo"] = 171,
2506 ["not"] = 172,
2507 ["shy"] = 173,
2508 ["reg"] = 174,
2509 ["circledR"] = 174,
2510 ["REG"] = 174,
2511 ["macr"] = 175,
2512 ["OverBar"] = 175,
2513 ["strns"] = 175,
2514 ["deg"] = 176,
2515 ["plusmn"] = 177,
2516 ["pm"] = 177,
2517 ["PlusMinus"] = 177,
2518 ["sup2"] = 178,
2519 ["sup3"] = 179,
2520 ["acute"] = 180,
2521 ["DiacriticalAcute"] = 180,
2522 ["micro"] = 181,
2523 ["para"] = 182,
2524 ["middot"] = 183,
2525 ["centerdot"] = 183,
2526 ["CenterDot"] = 183,
2527 ["cedil"] = 184,
2528 ["Cedilla"] = 184,
2529 ["sup1"] = 185,
2530 ["ordm"] = 186,
2531 ["raquo"] = 187,
2532 ["frac14"] = 188,
2533 ["frac12"] = 189,
2534 ["half"] = 189,
2535 ["frac34"] = 190,
2536 ["iquest"] = 191,
2537 ["Agrave"] = 192,
2538 ["Aacute"] = 193,
2539 ["Acirc"] = 194,
2540 ["Atilde"] = 195,
2541 ["Auml"] = 196,

```

2542 ["Aring"] = 197,  
 2543 ["AElig"] = 198,  
 2544 ["Ccedil"] = 199,  
 2545 ["Egrave"] = 200,  
 2546 ["Eacute"] = 201,  
 2547 ["Ecirc"] = 202,  
 2548 ["Euml"] = 203,  
 2549 ["Igrave"] = 204,  
 2550 ["Iacute"] = 205,  
 2551 ["Icirc"] = 206,  
 2552 ["Iuml"] = 207,  
 2553 ["ETH"] = 208,  
 2554 ["Ntilde"] = 209,  
 2555 ["Ograve"] = 210,  
 2556 ["Oacute"] = 211,  
 2557 ["Ocirc"] = 212,  
 2558 ["Otilde"] = 213,  
 2559 ["Ouml"] = 214,  
 2560 ["times"] = 215,  
 2561 ["Oslash"] = 216,  
 2562 ["Ugrave"] = 217,  
 2563 ["Uacute"] = 218,  
 2564 ["Ucirc"] = 219,  
 2565 ["Uuml"] = 220,  
 2566 ["Yacute"] = 221,  
 2567 ["THORN"] = 222,  
 2568 ["szlig"] = 223,  
 2569 ["agrave"] = 224,  
 2570 ["aacute"] = 225,  
 2571 ["acirc"] = 226,  
 2572 ["atilde"] = 227,  
 2573 ["auml"] = 228,  
 2574 ["aring"] = 229,  
 2575 ["aelig"] = 230,  
 2576 ["ccedil"] = 231,  
 2577 ["egrave"] = 232,  
 2578 ["eacute"] = 233,  
 2579 ["ecirc"] = 234,  
 2580 ["euml"] = 235,  
 2581 ["igrave"] = 236,  
 2582 ["iacute"] = 237,  
 2583 ["icirc"] = 238,  
 2584 ["iuml"] = 239,  
 2585 ["eth"] = 240,  
 2586 ["ntilde"] = 241,  
 2587 ["ograve"] = 242,  
 2588 ["oacute"] = 243,

```

2589 ["ocirc"] = 244,
2590 ["otilde"] = 245,
2591 ["ouml"] = 246,
2592 ["divide"] = 247,
2593 ["div"] = 247,
2594 ["oslash"] = 248,
2595 ["ugrave"] = 249,
2596 ["uacute"] = 250,
2597 ["ucirc"] = 251,
2598 ["uuml"] = 252,
2599 ["yacute"] = 253,
2600 ["thorn"] = 254,
2601 ["yuml"] = 255,
2602 ["Amacr"] = 256,
2603 ["amacr"] = 257,
2604 ["Abreve"] = 258,
2605 ["abreve"] = 259,
2606 ["Aogon"] = 260,
2607 ["aogon"] = 261,
2608 ["Cacute"] = 262,
2609 ["cacute"] = 263,
2610 ["Ccirc"] = 264,
2611 ["ccirc"] = 265,
2612 ["Cdot"] = 266,
2613 ["cdot"] = 267,
2614 ["Ccaron"] = 268,
2615 ["ccaron"] = 269,
2616 ["Dcaron"] = 270,
2617 ["dcaron"] = 271,
2618 ["Dstrok"] = 272,
2619 ["dstrok"] = 273,
2620 ["Emacr"] = 274,
2621 ["emacr"] = 275,
2622 ["Edot"] = 278,
2623 ["edot"] = 279,
2624 ["Eogon"] = 280,
2625 ["eogon"] = 281,
2626 ["Ecaron"] = 282,
2627 ["ecaron"] = 283,
2628 ["Gcirc"] = 284,
2629 ["gcirc"] = 285,
2630 ["Gbreve"] = 286,
2631 ["gbreve"] = 287,
2632 ["Gdot"] = 288,
2633 ["gdot"] = 289,
2634 ["Gcedil"] = 290,
2635 ["Hcirc"] = 292,

```

```

2636 ["hcirc"] = 293,
2637 ["Hstrok"] = 294,
2638 ["hstrok"] = 295,
2639 ["Itilde"] = 296,
2640 ["itilde"] = 297,
2641 ["Imacr"] = 298,
2642 ["imacr"] = 299,
2643 ["Iogon"] = 302,
2644 ["iogon"] = 303,
2645 ["Idot"] = 304,
2646 ["imath"] = 305,
2647 ["inodot"] = 305,
2648 ["IJlig"] = 306,
2649 ["ijlig"] = 307,
2650 ["Jcirc"] = 308,
2651 ["jcirc"] = 309,
2652 ["Kcedil"] = 310,
2653 ["kcedil"] = 311,
2654 ["kgreen"] = 312,
2655 ["Lacute"] = 313,
2656 ["lacute"] = 314,
2657 ["Lcedil"] = 315,
2658 ["lcedil"] = 316,
2659 ["Lcaron"] = 317,
2660 ["lcaron"] = 318,
2661 ["Lmidot"] = 319,
2662 ["lmidot"] = 320,
2663 ["Lstrok"] = 321,
2664 ["lstrok"] = 322,
2665 ["Nacute"] = 323,
2666 ["nacute"] = 324,
2667 ["Ncedil"] = 325,
2668 ["ncedil"] = 326,
2669 ["Ncaron"] = 327,
2670 ["ncaron"] = 328,
2671 ["napos"] = 329,
2672 ["ENG"] = 330,
2673 ["eng"] = 331,
2674 ["Omacr"] = 332,
2675 ["omacr"] = 333,
2676 ["Odblac"] = 336,
2677 ["odblac"] = 337,
2678 ["OElig"] = 338,
2679 ["oelig"] = 339,
2680 ["Racute"] = 340,
2681 ["racute"] = 341,
2682 ["Rcedil"] = 342,

```

2683 ["rcedil"] = 343,  
 2684 ["Rcaron"] = 344,  
 2685 ["rcaron"] = 345,  
 2686 ["Sacute"] = 346,  
 2687 ["sacute"] = 347,  
 2688 ["Scirc"] = 348,  
 2689 ["scirc"] = 349,  
 2690 ["Scedil"] = 350,  
 2691 ["scedil"] = 351,  
 2692 ["Scaron"] = 352,  
 2693 ["scaron"] = 353,  
 2694 ["Tcedil"] = 354,  
 2695 ["tcedil"] = 355,  
 2696 ["Tcaron"] = 356,  
 2697 ["tcaron"] = 357,  
 2698 ["Tstrok"] = 358,  
 2699 ["tstrok"] = 359,  
 2700 ["Utilde"] = 360,  
 2701 ["utilde"] = 361,  
 2702 ["Umacr"] = 362,  
 2703 ["umacr"] = 363,  
 2704 ["Ubreve"] = 364,  
 2705 ["ubreve"] = 365,  
 2706 ["Uring"] = 366,  
 2707 ["uring"] = 367,  
 2708 ["Udblac"] = 368,  
 2709 ["udblac"] = 369,  
 2710 ["Uogon"] = 370,  
 2711 ["uogon"] = 371,  
 2712 ["Wcirc"] = 372,  
 2713 ["wcirc"] = 373,  
 2714 ["Ycirc"] = 374,  
 2715 ["ycirc"] = 375,  
 2716 ["Yuml"] = 376,  
 2717 ["Zacute"] = 377,  
 2718 ["zacute"] = 378,  
 2719 ["Zdot"] = 379,  
 2720 ["zdot"] = 380,  
 2721 ["Zcaron"] = 381,  
 2722 ["zcaron"] = 382,  
 2723 ["fnof"] = 402,  
 2724 ["imped"] = 437,  
 2725 ["gacute"] = 501,  
 2726 ["jmath"] = 567,  
 2727 ["circ"] = 710,  
 2728 ["caron"] = 711,  
 2729 ["Hacek"] = 711,

2730 ["breve"] = 728,  
 2731 ["Breve"] = 728,  
 2732 ["dot"] = 729,  
 2733 ["DiacriticalDot"] = 729,  
 2734 ["ring"] = 730,  
 2735 ["ogon"] = 731,  
 2736 ["tilde"] = 732,  
 2737 ["DiacriticalTilde"] = 732,  
 2738 ["dblac"] = 733,  
 2739 ["DiacriticalDoubleAcute"] = 733,  
 2740 ["DownBreve"] = 785,  
 2741 ["UnderBar"] = 818,  
 2742 ["Alpha"] = 913,  
 2743 ["Beta"] = 914,  
 2744 ["Gamma"] = 915,  
 2745 ["Delta"] = 916,  
 2746 ["Epsilon"] = 917,  
 2747 ["Zeta"] = 918,  
 2748 ["Eta"] = 919,  
 2749 ["Theta"] = 920,  
 2750 ["Iota"] = 921,  
 2751 ["Kappa"] = 922,  
 2752 ["Lambda"] = 923,  
 2753 ["Mu"] = 924,  
 2754 ["Nu"] = 925,  
 2755 ["Xi"] = 926,  
 2756 ["Omicron"] = 927,  
 2757 ["Pi"] = 928,  
 2758 ["Rho"] = 929,  
 2759 ["Sigma"] = 931,  
 2760 ["Tau"] = 932,  
 2761 ["Upsilon"] = 933,  
 2762 ["Phi"] = 934,  
 2763 ["Chi"] = 935,  
 2764 ["Psi"] = 936,  
 2765 ["Omega"] = 937,  
 2766 ["alpha"] = 945,  
 2767 ["beta"] = 946,  
 2768 ["gamma"] = 947,  
 2769 ["delta"] = 948,  
 2770 ["epsiv"] = 949,  
 2771 ["varepsilon"] = 949,  
 2772 ["epsilon"] = 949,  
 2773 ["zeta"] = 950,  
 2774 ["eta"] = 951,  
 2775 ["theta"] = 952,  
 2776 ["iota"] = 953,



```

2777 ["kappa"] = 954,
2778 ["lambda"] = 955,
2779 ["mu"] = 956,
2780 ["nu"] = 957,
2781 ["xi"] = 958,
2782 ["omicron"] = 959,
2783 ["pi"] = 960,
2784 ["rho"] = 961,
2785 ["sigmav"] = 962,
2786 ["varsigma"] = 962,
2787 ["sigmaf"] = 962,
2788 ["sigma"] = 963,
2789 ["tau"] = 964,
2790 ["upsi"] = 965,
2791 ["upsilon"] = 965,
2792 ["phi"] = 966,
2793 ["phiv"] = 966,
2794 ["varphi"] = 966,
2795 ["chi"] = 967,
2796 ["psi"] = 968,
2797 ["omega"] = 969,
2798 ["thetav"] = 977,
2799 ["vartheta"] = 977,
2800 ["thetasym"] = 977,
2801 ["Upsi"] = 978,
2802 ["upsih"] = 978,
2803 ["straightphi"] = 981,
2804 ["piv"] = 982,
2805 ["varpi"] = 982,
2806 ["Gammad"] = 988,
2807 ["gammad"] = 989,
2808 ["digamma"] = 989,
2809 ["kappav"] = 1008,
2810 ["varkappa"] = 1008,
2811 ["rhov"] = 1009,
2812 ["varrho"] = 1009,
2813 ["epsi"] = 1013,
2814 ["straightepsilon"] = 1013,
2815 ["bepsi"] = 1014,
2816 ["backepsilon"] = 1014,
2817 ["IOcy"] = 1025,
2818 ["DJcy"] = 1026,
2819 ["GJcy"] = 1027,
2820 ["Jukcy"] = 1028,
2821 ["DScy"] = 1029,
2822 ["Iukcy"] = 1030,
2823 ["YIcy"] = 1031,

```

2824 ["Jsercy"] = 1032,  
2825 ["LJcy"] = 1033,  
2826 ["NJcy"] = 1034,  
2827 ["TSHcy"] = 1035,  
2828 ["KJcy"] = 1036,  
2829 ["Ubrcy"] = 1038,  
2830 ["DZcy"] = 1039,  
2831 ["Acy"] = 1040,  
2832 ["Bcy"] = 1041,  
2833 ["Vcy"] = 1042,  
2834 ["Gcy"] = 1043,  
2835 ["Dcy"] = 1044,  
2836 ["IEcy"] = 1045,  
2837 ["ZHcy"] = 1046,  
2838 ["Zcy"] = 1047,  
2839 ["Icy"] = 1048,  
2840 ["Jcy"] = 1049,  
2841 ["Kcy"] = 1050,  
2842 ["Lcy"] = 1051,  
2843 ["Mcy"] = 1052,  
2844 ["Ncy"] = 1053,  
2845 ["Ocy"] = 1054,  
2846 ["Pcy"] = 1055,  
2847 ["Rcy"] = 1056,  
2848 ["Scy"] = 1057,  
2849 ["Tcy"] = 1058,  
2850 ["Ucy"] = 1059,  
2851 ["Fcy"] = 1060,  
2852 ["KHcy"] = 1061,  
2853 ["TScy"] = 1062,  
2854 ["CHcy"] = 1063,  
2855 ["SHcy"] = 1064,  
2856 ["SHCHcy"] = 1065,  
2857 ["HARDcy"] = 1066,  
2858 ["Ycy"] = 1067,  
2859 ["SOFTcy"] = 1068,  
2860 ["Ecy"] = 1069,  
2861 ["YUcy"] = 1070,  
2862 ["YAcy"] = 1071,  
2863 ["acy"] = 1072,  
2864 ["bcy"] = 1073,  
2865 ["vcy"] = 1074,  
2866 ["gcy"] = 1075,  
2867 ["dcy"] = 1076,  
2868 ["iecy"] = 1077,  
2869 ["zhcy"] = 1078,  
2870 ["zcy"] = 1079,

```

2871 ["icy"] = 1080,
2872 ["jcy"] = 1081,
2873 ["kcy"] = 1082,
2874 ["lcy"] = 1083,
2875 ["mcy"] = 1084,
2876 ["ncy"] = 1085,
2877 ["ocy"] = 1086,
2878 ["pcy"] = 1087,
2879 ["rcy"] = 1088,
2880 ["scy"] = 1089,
2881 ["tcy"] = 1090,
2882 ["ucy"] = 1091,
2883 ["fcy"] = 1092,
2884 ["khcy"] = 1093,
2885 ["tscy"] = 1094,
2886 ["chcy"] = 1095,
2887 ["shcy"] = 1096,
2888 ["shchcy"] = 1097,
2889 ["hardcy"] = 1098,
2890 ["ycy"] = 1099,
2891 ["softcy"] = 1100,
2892 ["ecy"] = 1101,
2893 ["yucy"] = 1102,
2894 ["yacy"] = 1103,
2895 ["iocy"] = 1105,
2896 ["djcy"] = 1106,
2897 ["gjcy"] = 1107,
2898 ["jukcy"] = 1108,
2899 ["dscy"] = 1109,
2900 ["iukcy"] = 1110,
2901 ["yicy"] = 1111,
2902 ["jsercy"] = 1112,
2903 ["ljcy"] = 1113,
2904 ["njcy"] = 1114,
2905 ["tshcy"] = 1115,
2906 ["kjcy"] = 1116,
2907 ["ubrcy"] = 1118,
2908 ["dzcy"] = 1119,
2909 ["ensp"] = 8194,
2910 ["emsp"] = 8195,
2911 ["emsp13"] = 8196,
2912 ["emsp14"] = 8197,
2913 ["numsp"] = 8199,
2914 ["puncsp"] = 8200,
2915 ["thinsp"] = 8201,
2916 ["ThinSpace"] = 8201,
2917 ["hairsp"] = 8202,

```

```

2918 ["VeryThinSpace"] = 8202,
2919 ["ZeroWidthSpace"] = 8203,
2920 ["NegativeVeryThinSpace"] = 8203,
2921 ["NegativeThinSpace"] = 8203,
2922 ["NegativeMediumSpace"] = 8203,
2923 ["NegativeThickSpace"] = 8203,
2924 ["zwnj"] = 8204,
2925 ["zwj"] = 8205,
2926 ["lrm"] = 8206,
2927 ["rlm"] = 8207,
2928 ["hyphen"] = 8208,
2929 ["dash"] = 8208,
2930 ["ndash"] = 8211,
2931 ["mdash"] = 8212,
2932 ["horbar"] = 8213,
2933 ["Verbar"] = 8214,
2934 ["Vert"] = 8214,
2935 ["lsquo"] = 8216,
2936 ["OpenCurlyQuote"] = 8216,
2937 ["rsquo"] = 8217,
2938 ["rsquor"] = 8217,
2939 ["CloseCurlyQuote"] = 8217,
2940 ["lsquor"] = 8218,
2941 ["sbquo"] = 8218,
2942 ["ldquo"] = 8220,
2943 ["OpenCurlyDoubleQuote"] = 8220,
2944 ["rdquo"] = 8221,
2945 ["rdquor"] = 8221,
2946 ["CloseCurlyDoubleQuote"] = 8221,
2947 ["ldquor"] = 8222,
2948 ["bdquo"] = 8222,
2949 ["dagger"] = 8224,
2950 ["Dagger"] = 8225,
2951 ["ddagger"] = 8225,
2952 ["bull"] = 8226,
2953 ["bullet"] = 8226,
2954 ["nldr"] = 8229,
2955 ["hellip"] = 8230,
2956 ["mldr"] = 8230,
2957 ["permil"] = 8240,
2958 ["pertenk"] = 8241,
2959 ["prime"] = 8242,
2960 ["Prime"] = 8243,
2961 ["tprime"] = 8244,
2962 ["bprime"] = 8245,
2963 ["backprime"] = 8245,
2964 ["lsaquo"] = 8249,

```

```

2965 ["rsaquo"] = 8250,
2966 ["oline"] = 8254,
2967 ["caret"] = 8257,
2968 ["hybull"] = 8259,
2969 ["frasl"] = 8260,
2970 ["bsemi"] = 8271,
2971 ["qprime"] = 8279,
2972 ["MediumSpace"] = 8287,
2973 ["NoBreak"] = 8288,
2974 ["ApplyFunction"] = 8289,
2975 ["af"] = 8289,
2976 ["InvisibleTimes"] = 8290,
2977 ["it"] = 8290,
2978 ["InvisibleComma"] = 8291,
2979 ["ic"] = 8291,
2980 ["euro"] = 8364,
2981 ["tdot"] = 8411,
2982 ["TripleDot"] = 8411,
2983 ["DotDot"] = 8412,
2984 ["Copf"] = 8450,
2985 ["complexes"] = 8450,
2986 ["incare"] = 8453,
2987 ["gscr"] = 8458,
2988 ["hamilt"] = 8459,
2989 ["HilbertSpace"] = 8459,
2990 ["Hscr"] = 8459,
2991 ["Hfr"] = 8460,
2992 ["Poincareplane"] = 8460,
2993 ["quaternions"] = 8461,
2994 ["Hopf"] = 8461,
2995 ["planckh"] = 8462,
2996 ["planck"] = 8463,
2997 ["hbar"] = 8463,
2998 ["plankv"] = 8463,
2999 ["hslash"] = 8463,
3000 ["Iscr"] = 8464,
3001 ["imagline"] = 8464,
3002 ["image"] = 8465,
3003 ["Im"] = 8465,
3004 ["imagpart"] = 8465,
3005 ["Ifr"] = 8465,
3006 ["Lscr"] = 8466,
3007 ["lagran"] = 8466,
3008 ["Laplacetrif"] = 8466,
3009 ["ell"] = 8467,
3010 ["Nopf"] = 8469,
3011 ["naturals"] = 8469,

```

```

3012 ["numero"] = 8470,
3013 ["copysr"] = 8471,
3014 ["weierp"] = 8472,
3015 ["wp"] = 8472,
3016 ["Popf"] = 8473,
3017 ["primes"] = 8473,
3018 ["rationals"] = 8474,
3019 ["Qopf"] = 8474,
3020 ["Rscr"] = 8475,
3021 ["realine"] = 8475,
3022 ["real"] = 8476,
3023 ["Re"] = 8476,
3024 ["realpart"] = 8476,
3025 ["Rfr"] = 8476,
3026 ["reals"] = 8477,
3027 ["Ropf"] = 8477,
3028 ["rx"] = 8478,
3029 ["trade"] = 8482,
3030 ["TRADE"] = 8482,
3031 ["integers"] = 8484,
3032 ["Zopf"] = 8484,
3033 ["ohm"] = 8486,
3034 ["mho"] = 8487,
3035 ["Zfr"] = 8488,
3036 ["zeetrf"] = 8488,
3037 ["iiota"] = 8489,
3038 ["angst"] = 8491,
3039 ["bernou"] = 8492,
3040 ["Bernoullis"] = 8492,
3041 ["Bscr"] = 8492,
3042 ["Cfr"] = 8493,
3043 ["Cayleys"] = 8493,
3044 ["escr"] = 8495,
3045 ["Escr"] = 8496,
3046 ["expectation"] = 8496,
3047 ["Fscr"] = 8497,
3048 ["Fouriertrf"] = 8497,
3049 ["phmmat"] = 8499,
3050 ["Mellintrf"] = 8499,
3051 ["Mscr"] = 8499,
3052 ["order"] = 8500,
3053 ["orderof"] = 8500,
3054 ["oscr"] = 8500,
3055 ["alefsym"] = 8501,
3056 ["aleph"] = 8501,
3057 ["beth"] = 8502,
3058 ["gimel"] = 8503,

```

```

3059 ["daleth"] = 8504,
3060 ["CapitalDifferentialD"] = 8517,
3061 ["DD"] = 8517,
3062 ["DifferentialD"] = 8518,
3063 ["dd"] = 8518,
3064 ["ExponentialE"] = 8519,
3065 ["exponentiale"] = 8519,
3066 ["ee"] = 8519,
3067 ["ImaginaryI"] = 8520,
3068 ["ii"] = 8520,
3069 ["frac13"] = 8531,
3070 ["frac23"] = 8532,
3071 ["frac15"] = 8533,
3072 ["frac25"] = 8534,
3073 ["frac35"] = 8535,
3074 ["frac45"] = 8536,
3075 ["frac16"] = 8537,
3076 ["frac56"] = 8538,
3077 ["frac18"] = 8539,
3078 ["frac38"] = 8540,
3079 ["frac58"] = 8541,
3080 ["frac78"] = 8542,
3081 ["larr"] = 8592,
3082 ["leftarrow"] = 8592,
3083 ["LeftArrow"] = 8592,
3084 ["slarr"] = 8592,
3085 ["ShortLeftArrow"] = 8592,
3086 ["uarr"] = 8593,
3087 ["uparrow"] = 8593,
3088 ["UpArrow"] = 8593,
3089 ["ShortUpArrow"] = 8593,
3090 ["rarr"] = 8594,
3091 ["rightarrow"] = 8594,
3092 ["RightArrow"] = 8594,
3093 ["srarr"] = 8594,
3094 ["ShortRightArrow"] = 8594,
3095 ["darr"] = 8595,
3096 ["downarrow"] = 8595,
3097 ["DownArrow"] = 8595,
3098 ["ShortDownArrow"] = 8595,
3099 ["harr"] = 8596,
3100 ["leftrightarrow"] = 8596,
3101 ["LeftRightArrow"] = 8596,
3102 ["varr"] = 8597,
3103 ["updownarrow"] = 8597,
3104 ["UpDownArrow"] = 8597,
3105 ["nwarr"] = 8598,

```

```

3106 ["UpperLeftArrow"] = 8598,
3107 ["nwarrow"] = 8598,
3108 ["nearr"] = 8599,
3109 ["UpperRightArrow"] = 8599,
3110 ["nearrow"] = 8599,
3111 ["searr"] = 8600,
3112 ["searrow"] = 8600,
3113 ["LowerRightArrow"] = 8600,
3114 ["swarr"] = 8601,
3115 ["swarrow"] = 8601,
3116 ["LowerLeftArrow"] = 8601,
3117 ["nlarr"] = 8602,
3118 ["nleftarrow"] = 8602,
3119 ["nrarr"] = 8603,
3120 ["nrightarrow"] = 8603,
3121 ["rarrw"] = 8605,
3122 ["rightsquigarrow"] = 8605,
3123 ["Larr"] = 8606,
3124 ["twoheadleftarrow"] = 8606,
3125 ["Uarr"] = 8607,
3126 ["Rarr"] = 8608,
3127 ["twoheadrightarrow"] = 8608,
3128 ["Darr"] = 8609,
3129 ["larrtl"] = 8610,
3130 ["leftarrowtail"] = 8610,
3131 ["rarrtl"] = 8611,
3132 ["rightarrowtail"] = 8611,
3133 ["LeftTeeArrow"] = 8612,
3134 ["mapstoleft"] = 8612,
3135 ["UpTeeArrow"] = 8613,
3136 ["mapstoup"] = 8613,
3137 ["map"] = 8614,
3138 ["RightTeeArrow"] = 8614,
3139 ["mapsto"] = 8614,
3140 ["DownTeeArrow"] = 8615,
3141 ["mapstodown"] = 8615,
3142 ["larrhk"] = 8617,
3143 ["hookleftarrow"] = 8617,
3144 ["rarrhk"] = 8618,
3145 ["hookrightarrow"] = 8618,
3146 ["larrlp"] = 8619,
3147 ["looparrowleft"] = 8619,
3148 ["rarrlp"] = 8620,
3149 ["looparrowright"] = 8620,
3150 ["harrw"] = 8621,
3151 ["leftrightsquigarrow"] = 8621,
3152 ["nharr"] = 8622,

```



```

3153 ["nleftrightarrow"] = 8622,
3154 ["lsh"] = 8624,
3155 ["Lsh"] = 8624,
3156 ["rsh"] = 8625,
3157 ["Rsh"] = 8625,
3158 ["ldsh"] = 8626,
3159 ["rdsh"] = 8627,
3160 ["crarr"] = 8629,
3161 ["cularr"] = 8630,
3162 ["curvearrowleft"] = 8630,
3163 ["curarr"] = 8631,
3164 ["curvearrowright"] = 8631,
3165 ["olarr"] = 8634,
3166 ["circlearrowleft"] = 8634,
3167 ["orarr"] = 8635,
3168 ["circlearrowright"] = 8635,
3169 ["lharu"] = 8636,
3170 ["LeftVector"] = 8636,
3171 ["leftharpoonup"] = 8636,
3172 ["lhard"] = 8637,
3173 ["leftharpoondown"] = 8637,
3174 ["DownLeftVector"] = 8637,
3175 ["uharr"] = 8638,
3176 ["upharpoonright"] = 8638,
3177 ["RightUpVector"] = 8638,
3178 ["uharl"] = 8639,
3179 ["upharpoonleft"] = 8639,
3180 ["LeftUpVector"] = 8639,
3181 ["rharu"] = 8640,
3182 ["RightVector"] = 8640,
3183 ["rightharpoonup"] = 8640,
3184 ["rhard"] = 8641,
3185 ["rightharpoondown"] = 8641,
3186 ["DownRightVector"] = 8641,
3187 ["dharr"] = 8642,
3188 ["RightDownVector"] = 8642,
3189 ["downharpoonright"] = 8642,
3190 ["dharl"] = 8643,
3191 ["LeftDownVector"] = 8643,
3192 ["downharpoonleft"] = 8643,
3193 ["rlarr"] = 8644,
3194 ["rightleftarrows"] = 8644,
3195 ["RightArrowLeftArrow"] = 8644,
3196 ["udarr"] = 8645,
3197 ["UpArrowDownArrow"] = 8645,
3198 ["lrarr"] = 8646,
3199 ["leftrightarrows"] = 8646,

```

```

3200 ["LeftArrowRightArrow"] = 8646,
3201 ["llarr"] = 8647,
3202 ["leftleftarrows"] = 8647,
3203 ["uuarr"] = 8648,
3204 ["upuparrows"] = 8648,
3205 ["rrarr"] = 8649,
3206 ["rightrightarrows"] = 8649,
3207 ["ddarr"] = 8650,
3208 ["downdownarrows"] = 8650,
3209 ["lrhar"] = 8651,
3210 ["ReverseEquilibrium"] = 8651,
3211 ["leftrightharpoons"] = 8651,
3212 ["rlhar"] = 8652,
3213 ["rightleftharpoons"] = 8652,
3214 ["Equilibrium"] = 8652,
3215 ["nlArr"] = 8653,
3216 ["nLeftarrow"] = 8653,
3217 ["nhArr"] = 8654,
3218 ["nLeftrightarrow"] = 8654,
3219 ["nrArr"] = 8655,
3220 ["nRightarrow"] = 8655,
3221 ["lArr"] = 8656,
3222 ["Leftarrow"] = 8656,
3223 ["DoubleLeftArrow"] = 8656,
3224 ["uArr"] = 8657,
3225 ["Uparrow"] = 8657,
3226 ["DoubleUpArrow"] = 8657,
3227 ["rArr"] = 8658,
3228 ["Rightarrow"] = 8658,
3229 ["Implies"] = 8658,
3230 ["DoubleRightArrow"] = 8658,
3231 ["dArr"] = 8659,
3232 ["Downarrow"] = 8659,
3233 ["DoubleDownArrow"] = 8659,
3234 ["hArr"] = 8660,
3235 ["Leftrightarrow"] = 8660,
3236 ["DoubleLeftRightArrow"] = 8660,
3237 ["iff"] = 8660,
3238 ["vArr"] = 8661,
3239 ["Updownarrow"] = 8661,
3240 ["DoubleUpDownArrow"] = 8661,
3241 ["nwArr"] = 8662,
3242 ["neArr"] = 8663,
3243 ["seArr"] = 8664,
3244 ["swArr"] = 8665,
3245 ["lAarr"] = 8666,
3246 ["Lleftarrow"] = 8666,

```

```

3247 ["rAarr"] = 8667,
3248 ["Rrightarrow"] = 8667,
3249 ["zigrarr"] = 8669,
3250 ["larrb"] = 8676,
3251 ["LeftArrowBar"] = 8676,
3252 ["rarrb"] = 8677,
3253 ["RightArrowBar"] = 8677,
3254 ["duarr"] = 8693,
3255 ["DownArrowUpArrow"] = 8693,
3256 ["loarr"] = 8701,
3257 ["roarr"] = 8702,
3258 ["hoarr"] = 8703,
3259 ["forall"] = 8704,
3260 ["ForAll"] = 8704,
3261 ["comp"] = 8705,
3262 ["complement"] = 8705,
3263 ["part"] = 8706,
3264 ["PartialD"] = 8706,
3265 ["exist"] = 8707,
3266 ["Exists"] = 8707,
3267 ["nexist"] = 8708,
3268 ["NotExists"] = 8708,
3269 ["nexists"] = 8708,
3270 ["empty"] = 8709,
3271 ["emptyset"] = 8709,
3272 ["emptyv"] = 8709,
3273 ["varnothing"] = 8709,
3274 ["nabla"] = 8711,
3275 ["Del"] = 8711,
3276 ["isin"] = 8712,
3277 ["isinv"] = 8712,
3278 ["Element"] = 8712,
3279 ["in"] = 8712,
3280 ["notin"] = 8713,
3281 ["NotElement"] = 8713,
3282 ["notinva"] = 8713,
3283 ["niv"] = 8715,
3284 ["ReverseElement"] = 8715,
3285 ["ni"] = 8715,
3286 ["SuchThat"] = 8715,
3287 ["notni"] = 8716,
3288 ["notniva"] = 8716,
3289 ["NotReverseElement"] = 8716,
3290 ["prod"] = 8719,
3291 ["Product"] = 8719,
3292 ["coprod"] = 8720,
3293 ["Coproduct"] = 8720,

```

```

3294 ["sum"] = 8721,
3295 ["Sum"] = 8721,
3296 ["minus"] = 8722,
3297 ["mnplus"] = 8723,
3298 ["mp"] = 8723,
3299 ["MinusPlus"] = 8723,
3300 ["plusdo"] = 8724,
3301 ["dotplus"] = 8724,
3302 ["setmn"] = 8726,
3303 ["setminus"] = 8726,
3304 ["Backslash"] = 8726,
3305 ["ssetmn"] = 8726,
3306 ["smallsetminus"] = 8726,
3307 ["lowast"] = 8727,
3308 ["compfn"] = 8728,
3309 ["SmallCircle"] = 8728,
3310 ["radic"] = 8730,
3311 ["Sqrt"] = 8730,
3312 ["prop"] = 8733,
3313 ["propto"] = 8733,
3314 ["Proportional"] = 8733,
3315 ["vprop"] = 8733,
3316 ["varpropto"] = 8733,
3317 ["infin"] = 8734,
3318 ["angrt"] = 8735,
3319 ["ang"] = 8736,
3320 ["angle"] = 8736,
3321 ["angmsd"] = 8737,
3322 ["measuredangle"] = 8737,
3323 ["angsph"] = 8738,
3324 ["mid"] = 8739,
3325 ["VerticalBar"] = 8739,
3326 ["smid"] = 8739,
3327 ["shortmid"] = 8739,
3328 ["nmid"] = 8740,
3329 ["NotVerticalBar"] = 8740,
3330 ["nsmid"] = 8740,
3331 ["nshortmid"] = 8740,
3332 ["par"] = 8741,
3333 ["parallel"] = 8741,
3334 ["DoubleVerticalBar"] = 8741,
3335 ["spar"] = 8741,
3336 ["shortparallel"] = 8741,
3337 ["npar"] = 8742,
3338 ["nparallel"] = 8742,
3339 ["NotDoubleVerticalBar"] = 8742,
3340 ["nspar"] = 8742,

```

```

3341 ["nshortparallel"] = 8742,
3342 ["and"] = 8743,
3343 ["wedge"] = 8743,
3344 ["or"] = 8744,
3345 ["vee"] = 8744,
3346 ["cap"] = 8745,
3347 ["cup"] = 8746,
3348 ["int"] = 8747,
3349 ["Integral"] = 8747,
3350 ["Int"] = 8748,
3351 ["tint"] = 8749,
3352 ["iiint"] = 8749,
3353 ["conint"] = 8750,
3354 ["oint"] = 8750,
3355 ["ContourIntegral"] = 8750,
3356 ["Conint"] = 8751,
3357 ["DoubleContourIntegral"] = 8751,
3358 ["Cconint"] = 8752,
3359 ["cwint"] = 8753,
3360 ["cwconint"] = 8754,
3361 ["ClockwiseContourIntegral"] = 8754,
3362 ["awconint"] = 8755,
3363 ["CounterClockwiseContourIntegral"] = 8755,
3364 ["there4"] = 8756,
3365 ["therefore"] = 8756,
3366 ["Therefore"] = 8756,
3367 ["because"] = 8757,
3368 ["because"] = 8757,
3369 ["Because"] = 8757,
3370 ["ratio"] = 8758,
3371 ["Colon"] = 8759,
3372 ["Proportion"] = 8759,
3373 ["minusd"] = 8760,
3374 ["dotminus"] = 8760,
3375 ["mDDot"] = 8762,
3376 ["homtht"] = 8763,
3377 ["sim"] = 8764,
3378 ["Tilde"] = 8764,
3379 ["thksim"] = 8764,
3380 ["thicksim"] = 8764,
3381 ["bsim"] = 8765,
3382 ["backsim"] = 8765,
3383 ["ac"] = 8766,
3384 ["mstpos"] = 8766,
3385 ["acd"] = 8767,
3386 ["wreath"] = 8768,
3387 ["VerticalTilde"] = 8768,

```

```

3388 ["wr"] = 8768,
3389 ["nsim"] = 8769,
3390 ["NotTilde"] = 8769,
3391 ["esim"] = 8770,
3392 ["EqualTilde"] = 8770,
3393 ["eqsim"] = 8770,
3394 ["sime"] = 8771,
3395 ["TildeEqual"] = 8771,
3396 ["simeq"] = 8771,
3397 ["nsime"] = 8772,
3398 ["nsimeq"] = 8772,
3399 ["NotTildeEqual"] = 8772,
3400 ["cong"] = 8773,
3401 ["TildeFullEqual"] = 8773,
3402 ["simne"] = 8774,
3403 ["ncong"] = 8775,
3404 ["NotTildeFullEqual"] = 8775,
3405 ["asymp"] = 8776,
3406 ["ap"] = 8776,
3407 ["TildeTilde"] = 8776,
3408 ["approx"] = 8776,
3409 ["thkap"] = 8776,
3410 ["thickapprox"] = 8776,
3411 ["nap"] = 8777,
3412 ["NotTildeTilde"] = 8777,
3413 ["naprox"] = 8777,
3414 ["ape"] = 8778,
3415 ["approxpeq"] = 8778,
3416 ["apid"] = 8779,
3417 ["bcong"] = 8780,
3418 ["backcong"] = 8780,
3419 ["asympeq"] = 8781,
3420 ["CupCap"] = 8781,
3421 ["bump"] = 8782,
3422 ["HumpDownHump"] = 8782,
3423 ["Bumpeq"] = 8782,
3424 ["bumpe"] = 8783,
3425 ["HumpEqual"] = 8783,
3426 ["bumpeq"] = 8783,
3427 ["esdot"] = 8784,
3428 ["DotEqual"] = 8784,
3429 ["doteq"] = 8784,
3430 ["eDot"] = 8785,
3431 ["doteqdot"] = 8785,
3432 ["efDot"] = 8786,
3433 ["fallingdotseq"] = 8786,
3434 ["erDot"] = 8787,

```

```

3435 ["risingdotseq"] = 8787,
3436 ["colone"] = 8788,
3437 ["coloneq"] = 8788,
3438 ["Assign"] = 8788,
3439 ["ecolon"] = 8789,
3440 ["eqcolon"] = 8789,
3441 ["ecir"] = 8790,
3442 ["eqcirc"] = 8790,
3443 ["cire"] = 8791,
3444 ["circeq"] = 8791,
3445 ["wedgeq"] = 8793,
3446 ["veeeq"] = 8794,
3447 ["trie"] = 8796,
3448 ["triangleq"] = 8796,
3449 ["equest"] = 8799,
3450 ["questeq"] = 8799,
3451 ["ne"] = 8800,
3452 ["NotEqual"] = 8800,
3453 ["equiv"] = 8801,
3454 ["Congruent"] = 8801,
3455 ["nequiv"] = 8802,
3456 ["NotCongruent"] = 8802,
3457 ["le"] = 8804,
3458 ["leq"] = 8804,
3459 ["ge"] = 8805,
3460 ["GreaterEqual"] = 8805,
3461 ["geq"] = 8805,
3462 ["lE"] = 8806,
3463 ["LessFullEqual"] = 8806,
3464 ["leqq"] = 8806,
3465 ["gE"] = 8807,
3466 ["GreaterFullEqual"] = 8807,
3467 ["geqq"] = 8807,
3468 ["lnE"] = 8808,
3469 ["lneqq"] = 8808,
3470 ["gnE"] = 8809,
3471 ["gneqq"] = 8809,
3472 ["Lt"] = 8810,
3473 ["NestedLessLess"] = 8810,
3474 ["ll"] = 8810,
3475 ["Gt"] = 8811,
3476 ["NestedGreaterGreater"] = 8811,
3477 ["gg"] = 8811,
3478 ["twixt"] = 8812,
3479 ["between"] = 8812,
3480 ["NotCupCap"] = 8813,
3481 ["nlt"] = 8814,

```

```

3482 ["NotLess"] = 8814,
3483 ["nless"] = 8814,
3484 ["ngt"] = 8815,
3485 ["NotGreater"] = 8815,
3486 ["ngtr"] = 8815,
3487 ["nle"] = 8816,
3488 ["NotLessEqual"] = 8816,
3489 ["nleq"] = 8816,
3490 ["nge"] = 8817,
3491 ["NotGreaterEqual"] = 8817,
3492 ["ngeq"] = 8817,
3493 ["lsim"] = 8818,
3494 ["LessTilde"] = 8818,
3495 ["lesssim"] = 8818,
3496 ["gsim"] = 8819,
3497 ["gtrsim"] = 8819,
3498 ["GreaterTilde"] = 8819,
3499 ["nlsim"] = 8820,
3500 ["NotLessTilde"] = 8820,
3501 ["ngsim"] = 8821,
3502 ["NotGreaterTilde"] = 8821,
3503 ["lg"] = 8822,
3504 ["lessgtr"] = 8822,
3505 ["LessGreater"] = 8822,
3506 ["gl"] = 8823,
3507 ["gtrless"] = 8823,
3508 ["GreaterLess"] = 8823,
3509 ["ntlg"] = 8824,
3510 ["NotLessGreater"] = 8824,
3511 ["ntgl"] = 8825,
3512 ["NotGreaterLess"] = 8825,
3513 ["pr"] = 8826,
3514 ["Precedes"] = 8826,
3515 ["prec"] = 8826,
3516 ["sc"] = 8827,
3517 ["Succeeds"] = 8827,
3518 ["succ"] = 8827,
3519 ["prcue"] = 8828,
3520 ["PrecedesSlantEqual"] = 8828,
3521 ["preccurlyeq"] = 8828,
3522 ["sccue"] = 8829,
3523 ["SucceedsSlantEqual"] = 8829,
3524 ["succcurlyeq"] = 8829,
3525 ["prsim"] = 8830,
3526 ["precsim"] = 8830,
3527 ["PrecedesTilde"] = 8830,
3528 ["scsim"] = 8831,

```



```

3529 ["succsim"] = 8831,
3530 ["SucceedsTilde"] = 8831,
3531 ["npr"] = 8832,
3532 ["nprec"] = 8832,
3533 ["NotPrecedes"] = 8832,
3534 ["nsc"] = 8833,
3535 ["nsucc"] = 8833,
3536 ["NotSucceeds"] = 8833,
3537 ["sub"] = 8834,
3538 ["subset"] = 8834,
3539 ["sup"] = 8835,
3540 ["supset"] = 8835,
3541 ["Superset"] = 8835,
3542 ["nsub"] = 8836,
3543 ["nsup"] = 8837,
3544 ["sube"] = 8838,
3545 ["SubsetEqual"] = 8838,
3546 ["subseteq"] = 8838,
3547 ["supe"] = 8839,
3548 ["supseteq"] = 8839,
3549 ["SupersetEqual"] = 8839,
3550 ["nsube"] = 8840,
3551 ["nsubseteq"] = 8840,
3552 ["NotSubsetEqual"] = 8840,
3553 ["nsupe"] = 8841,
3554 ["nsupseteq"] = 8841,
3555 ["NotSupersetEqual"] = 8841,
3556 ["subne"] = 8842,
3557 ["subsetneq"] = 8842,
3558 ["supne"] = 8843,
3559 ["supsetneq"] = 8843,
3560 ["cupdot"] = 8845,
3561 ["uplus"] = 8846,
3562 ["UnionPlus"] = 8846,
3563 ["sqsub"] = 8847,
3564 ["SquareSubset"] = 8847,
3565 ["sqsubset"] = 8847,
3566 ["sqsup"] = 8848,
3567 ["SquareSuperset"] = 8848,
3568 ["sqsupset"] = 8848,
3569 ["sqsube"] = 8849,
3570 ["SquareSubsetEqual"] = 8849,
3571 ["sqsubseteq"] = 8849,
3572 ["sqsupe"] = 8850,
3573 ["SquareSupersetEqual"] = 8850,
3574 ["sqsupseteq"] = 8850,
3575 ["sqcap"] = 8851,

```

```

3576 ["SquareIntersection"] = 8851,
3577 ["sqcup"] = 8852,
3578 ["SquareUnion"] = 8852,
3579 ["oplus"] = 8853,
3580 ["CirclePlus"] = 8853,
3581 ["ominus"] = 8854,
3582 ["CircleMinus"] = 8854,
3583 ["otimes"] = 8855,
3584 ["CircleTimes"] = 8855,
3585 ["osol"] = 8856,
3586 ["odot"] = 8857,
3587 ["CircleDot"] = 8857,
3588 ["ocir"] = 8858,
3589 ["circledcirc"] = 8858,
3590 ["oast"] = 8859,
3591 ["circledast"] = 8859,
3592 ["odash"] = 8861,
3593 ["circleddash"] = 8861,
3594 ["plusb"] = 8862,
3595 ["boxplus"] = 8862,
3596 ["minusb"] = 8863,
3597 ["boxminus"] = 8863,
3598 ["timesb"] = 8864,
3599 ["boxtimes"] = 8864,
3600 ["sdotb"] = 8865,
3601 ["dotsquare"] = 8865,
3602 ["vdash"] = 8866,
3603 ["RightTee"] = 8866,
3604 ["dashv"] = 8867,
3605 ["LeftTee"] = 8867,
3606 ["top"] = 8868,
3607 ["DownTee"] = 8868,
3608 ["bottom"] = 8869,
3609 ["bot"] = 8869,
3610 ["perp"] = 8869,
3611 ["UpTee"] = 8869,
3612 ["models"] = 8871,
3613 ["vDash"] = 8872,
3614 ["DoubleRightTee"] = 8872,
3615 ["Vdash"] = 8873,
3616 ["Vvdash"] = 8874,
3617 ["VDash"] = 8875,
3618 ["nvdash"] = 8876,
3619 ["nvDash"] = 8877,
3620 ["nVdash"] = 8878,
3621 ["nVDash"] = 8879,
3622 ["prurel"] = 8880,

```

```

3623 ["vltri"] = 8882,
3624 ["vartriangleleft"] = 8882,
3625 ["LeftTriangle"] = 8882,
3626 ["vrtri"] = 8883,
3627 ["vartriangleright"] = 8883,
3628 ["RightTriangle"] = 8883,
3629 ["ltrie"] = 8884,
3630 ["trianglelefteq"] = 8884,
3631 ["LeftTriangleEqual"] = 8884,
3632 ["rtrie"] = 8885,
3633 ["trianglerighteq"] = 8885,
3634 ["RightTriangleEqual"] = 8885,
3635 ["origof"] = 8886,
3636 ["imof"] = 8887,
3637 ["mumap"] = 8888,
3638 ["multimap"] = 8888,
3639 ["hercon"] = 8889,
3640 ["intcal"] = 8890,
3641 ["intercal"] = 8890,
3642 ["veebar"] = 8891,
3643 ["barvee"] = 8893,
3644 ["angrtvb"] = 8894,
3645 ["lrtri"] = 8895,
3646 ["xwedge"] = 8896,
3647 ["Wedge"] = 8896,
3648 ["bigwedge"] = 8896,
3649 ["xvee"] = 8897,
3650 ["Vee"] = 8897,
3651 ["bigvee"] = 8897,
3652 ["xcap"] = 8898,
3653 ["Intersection"] = 8898,
3654 ["bigcap"] = 8898,
3655 ["xcup"] = 8899,
3656 ["Union"] = 8899,
3657 ["bigcup"] = 8899,
3658 ["diam"] = 8900,
3659 ["diamond"] = 8900,
3660 ["Diamond"] = 8900,
3661 ["sdot"] = 8901,
3662 ["sstarf"] = 8902,
3663 ["Star"] = 8902,
3664 ["divonx"] = 8903,
3665 ["divideontimes"] = 8903,
3666 ["bowtie"] = 8904,
3667 ["ltimes"] = 8905,
3668 ["rtimes"] = 8906,
3669 ["lthree"] = 8907,

```

```

3670 ["leftthreetimes"] = 8907,
3671 ["rthree"] = 8908,
3672 ["rightthreetimes"] = 8908,
3673 ["bsime"] = 8909,
3674 ["backsimeq"] = 8909,
3675 ["cuvee"] = 8910,
3676 ["curlyvee"] = 8910,
3677 ["cuwed"] = 8911,
3678 ["curlywedge"] = 8911,
3679 ["Sub"] = 8912,
3680 ["Subset"] = 8912,
3681 ["Sup"] = 8913,
3682 ["Supset"] = 8913,
3683 ["Cap"] = 8914,
3684 ["Cup"] = 8915,
3685 ["fork"] = 8916,
3686 ["pitchfork"] = 8916,
3687 ["epar"] = 8917,
3688 ["ltdot"] = 8918,
3689 ["lessdot"] = 8918,
3690 ["gtdot"] = 8919,
3691 ["gtrdot"] = 8919,
3692 ["Ll"] = 8920,
3693 ["Gg"] = 8921,
3694 ["ggg"] = 8921,
3695 ["leg"] = 8922,
3696 ["LessEqualGreater"] = 8922,
3697 ["lesseqgtr"] = 8922,
3698 ["gel"] = 8923,
3699 ["gtreqless"] = 8923,
3700 ["GreaterEqualLess"] = 8923,
3701 ["cuepr"] = 8926,
3702 ["curlyeqprec"] = 8926,
3703 ["cuesc"] = 8927,
3704 ["curlyeqsucc"] = 8927,
3705 ["nprcue"] = 8928,
3706 ["NotPrecedesSlantEqual"] = 8928,
3707 ["nsccue"] = 8929,
3708 ["NotSucceedsSlantEqual"] = 8929,
3709 ["nsqsube"] = 8930,
3710 ["NotSquareSubsetEqual"] = 8930,
3711 ["nsqsupe"] = 8931,
3712 ["NotSquareSupersetEqual"] = 8931,
3713 ["lnsim"] = 8934,
3714 ["gnsim"] = 8935,
3715 ["prnsim"] = 8936,
3716 ["precnsim"] = 8936,

```

```

3717 ["scnsim"] = 8937,
3718 ["succnsim"] = 8937,
3719 ["nltri"] = 8938,
3720 ["ntriangleleft"] = 8938,
3721 ["NotLeftTriangle"] = 8938,
3722 ["nrtri"] = 8939,
3723 ["ntriangleright"] = 8939,
3724 ["NotRightTriangle"] = 8939,
3725 ["nltrie"] = 8940,
3726 ["ntrianglelefteq"] = 8940,
3727 ["NotLeftTriangleEqual"] = 8940,
3728 ["nrtrie"] = 8941,
3729 ["ntrianglerighteq"] = 8941,
3730 ["NotRightTriangleEqual"] = 8941,
3731 ["vellip"] = 8942,
3732 ["ctdot"] = 8943,
3733 ["utdot"] = 8944,
3734 ["dtdot"] = 8945,
3735 ["disin"] = 8946,
3736 ["isinsv"] = 8947,
3737 ["isins"] = 8948,
3738 ["isindot"] = 8949,
3739 ["notinvc"] = 8950,
3740 ["notinvb"] = 8951,
3741 ["isinE"] = 8953,
3742 ["nisd"] = 8954,
3743 ["xnis"] = 8955,
3744 ["nis"] = 8956,
3745 ["notnivc"] = 8957,
3746 ["notnivb"] = 8958,
3747 ["barwed"] = 8965,
3748 ["barwedge"] = 8965,
3749 ["Barwed"] = 8966,
3750 ["doublebarwedge"] = 8966,
3751 ["lceil"] = 8968,
3752 ["LeftCeiling"] = 8968,
3753 ["rceil"] = 8969,
3754 ["RightCeiling"] = 8969,
3755 ["lfloor"] = 8970,
3756 ["LeftFloor"] = 8970,
3757 ["rfloor"] = 8971,
3758 ["RightFloor"] = 8971,
3759 ["drcrop"] = 8972,
3760 ["dlcrop"] = 8973,
3761 ["urcrop"] = 8974,
3762 ["ulcrop"] = 8975,
3763 ["bnot"] = 8976,

```

```

3764 ["proflin"] = 8978,
3765 ["profsurf"] = 8979,
3766 ["telrec"] = 8981,
3767 ["target"] = 8982,
3768 ["ulcorn"] = 8988,
3769 ["ulcorner"] = 8988,
3770 ["urcorn"] = 8989,
3771 ["urcorner"] = 8989,
3772 ["dlcorn"] = 8990,
3773 ["llcorner"] = 8990,
3774 ["drcorn"] = 8991,
3775 ["lrcorner"] = 8991,
3776 ["frown"] = 8994,
3777 ["sfrown"] = 8994,
3778 ["smile"] = 8995,
3779 ["ssmile"] = 8995,
3780 ["cylcty"] = 9005,
3781 ["profalar"] = 9006,
3782 ["topbot"] = 9014,
3783 ["ovbar"] = 9021,
3784 ["solbar"] = 9023,
3785 ["angzarr"] = 9084,
3786 ["lmoust"] = 9136,
3787 ["lmoustache"] = 9136,
3788 ["rmoust"] = 9137,
3789 ["rmoustache"] = 9137,
3790 ["tbrk"] = 9140,
3791 ["OverBracket"] = 9140,
3792 ["bbrk"] = 9141,
3793 ["UnderBracket"] = 9141,
3794 ["bbrktbrk"] = 9142,
3795 ["OverParenthesis"] = 9180,
3796 ["UnderParenthesis"] = 9181,
3797 ["OverBrace"] = 9182,
3798 ["UnderBrace"] = 9183,
3799 ["trpezium"] = 9186,
3800 ["elinters"] = 9191,
3801 ["blank"] = 9251,
3802 ["oS"] = 9416,
3803 ["circledS"] = 9416,
3804 ["boxh"] = 9472,
3805 ["HorizontalLine"] = 9472,
3806 ["boxv"] = 9474,
3807 ["boxdr"] = 9484,
3808 ["boxdl"] = 9488,
3809 ["boxur"] = 9492,
3810 ["boxul"] = 9496,

```

```

3811 ["boxvr"] = 9500,
3812 ["boxvl"] = 9508,
3813 ["boxhd"] = 9516,
3814 ["boxhu"] = 9524,
3815 ["boxvh"] = 9532,
3816 ["boxH"] = 9552,
3817 ["boxV"] = 9553,
3818 ["boxdR"] = 9554,
3819 ["boxDr"] = 9555,
3820 ["boxDR"] = 9556,
3821 ["boxdL"] = 9557,
3822 ["boxDl"] = 9558,
3823 ["boxDL"] = 9559,
3824 ["boxuR"] = 9560,
3825 ["boxUr"] = 9561,
3826 ["boxUR"] = 9562,
3827 ["boxuL"] = 9563,
3828 ["boxUl"] = 9564,
3829 ["boxUL"] = 9565,
3830 ["boxvR"] = 9566,
3831 ["boxVr"] = 9567,
3832 ["boxVR"] = 9568,
3833 ["boxvL"] = 9569,
3834 ["boxVl"] = 9570,
3835 ["boxVL"] = 9571,
3836 ["boxHd"] = 9572,
3837 ["boxhD"] = 9573,
3838 ["boxHD"] = 9574,
3839 ["boxHu"] = 9575,
3840 ["boxhU"] = 9576,
3841 ["boxHU"] = 9577,
3842 ["boxvH"] = 9578,
3843 ["boxVh"] = 9579,
3844 ["boxVH"] = 9580,
3845 ["uhblk"] = 9600,
3846 ["lhblk"] = 9604,
3847 ["block"] = 9608,
3848 ["blk14"] = 9617,
3849 ["blk12"] = 9618,
3850 ["blk34"] = 9619,
3851 ["squ"] = 9633,
3852 ["square"] = 9633,
3853 ["Square"] = 9633,
3854 ["squf"] = 9642,
3855 ["squarf"] = 9642,
3856 ["blacksquare"] = 9642,
3857 ["FilledVerySmallSquare"] = 9642,

```

```

3858 ["EmptyVerySmallSquare"] = 9643,
3859 ["rect"] = 9645,
3860 ["marker"] = 9646,
3861 ["fltns"] = 9649,
3862 ["xutri"] = 9651,
3863 ["bigtriangleup"] = 9651,
3864 ["utrif"] = 9652,
3865 ["blacktriangle"] = 9652,
3866 ["utri"] = 9653,
3867 ["triangle"] = 9653,
3868 ["rtrif"] = 9656,
3869 ["blacktriangleright"] = 9656,
3870 ["rtri"] = 9657,
3871 ["triangleright"] = 9657,
3872 ["xdtri"] = 9661,
3873 ["bigtriangledown"] = 9661,
3874 ["dtrif"] = 9662,
3875 ["blacktriangledown"] = 9662,
3876 ["dtri"] = 9663,
3877 ["triangledown"] = 9663,
3878 ["ltrif"] = 9666,
3879 ["blacktriangleleft"] = 9666,
3880 ["ltri"] = 9667,
3881 ["triangleleft"] = 9667,
3882 ["loz"] = 9674,
3883 ["lozenge"] = 9674,
3884 ["cir"] = 9675,
3885 ["tridot"] = 9708,
3886 ["xcirc"] = 9711,
3887 ["bigcirc"] = 9711,
3888 ["ultri"] = 9720,
3889 ["urtri"] = 9721,
3890 ["lltri"] = 9722,
3891 ["EmptySmallSquare"] = 9723,
3892 ["FilledSmallSquare"] = 9724,
3893 ["starf"] = 9733,
3894 ["bigstar"] = 9733,
3895 ["star"] = 9734,
3896 ["phone"] = 9742,
3897 ["female"] = 9792,
3898 ["male"] = 9794,
3899 ["spades"] = 9824,
3900 ["spadesuit"] = 9824,
3901 ["clubs"] = 9827,
3902 ["clubsuit"] = 9827,
3903 ["hearts"] = 9829,
3904 ["heartsuit"] = 9829,

```



```

3905 ["diams"] = 9830,
3906 ["diamondsuit"] = 9830,
3907 ["sung"] = 9834,
3908 ["flat"] = 9837,
3909 ["natur"] = 9838,
3910 ["natural"] = 9838,
3911 ["sharp"] = 9839,
3912 ["check"] = 10003,
3913 ["checkmark"] = 10003,
3914 ["cross"] = 10007,
3915 ["malt"] = 10016,
3916 ["maltese"] = 10016,
3917 ["sext"] = 10038,
3918 ["VerticalSeparator"] = 10072,
3919 ["lbbbrk"] = 10098,
3920 ["rbbbrk"] = 10099,
3921 ["lobrk"] = 10214,
3922 ["LeftDoubleBracket"] = 10214,
3923 ["robrk"] = 10215,
3924 ["RightDoubleBracket"] = 10215,
3925 ["lang"] = 10216,
3926 ["LeftAngleBracket"] = 10216,
3927 ["langle"] = 10216,
3928 ["rang"] = 10217,
3929 ["RightAngleBracket"] = 10217,
3930 ["rangle"] = 10217,
3931 ["Lang"] = 10218,
3932 ["Rang"] = 10219,
3933 ["loang"] = 10220,
3934 ["roang"] = 10221,
3935 ["xlarr"] = 10229,
3936 ["longleftarrow"] = 10229,
3937 ["LongLeftArrow"] = 10229,
3938 ["xrarr"] = 10230,
3939 ["longrightarrow"] = 10230,
3940 ["LongRightArrow"] = 10230,
3941 ["xharr"] = 10231,
3942 ["longleftrightharrow"] = 10231,
3943 ["LongLeftRightArrow"] = 10231,
3944 ["xlArr"] = 10232,
3945 ["Longleftarrow"] = 10232,
3946 ["DoubleLongLeftArrow"] = 10232,
3947 ["xrArr"] = 10233,
3948 ["Longrightarrow"] = 10233,
3949 ["DoubleLongRightArrow"] = 10233,
3950 ["xhArr"] = 10234,
3951 ["Longleftrightharrow"] = 10234,

```

```

3952 ["DoubleLongLeftRightArrow"] = 10234,
3953 ["xmap"] = 10236,
3954 ["longmapsto"] = 10236,
3955 ["dzigrarr"] = 10239,
3956 ["nvlArr"] = 10498,
3957 ["nvrArr"] = 10499,
3958 ["nvHarr"] = 10500,
3959 ["Map"] = 10501,
3960 ["lbarr"] = 10508,
3961 ["rbarr"] = 10509,
3962 ["bkarow"] = 10509,
3963 ["lBarr"] = 10510,
3964 ["rBarr"] = 10511,
3965 ["dbkarow"] = 10511,
3966 ["RBarr"] = 10512,
3967 ["drbkarow"] = 10512,
3968 ["DDottrahd"] = 10513,
3969 ["UpArrowBar"] = 10514,
3970 ["DownArrowBar"] = 10515,
3971 ["Rarrtl"] = 10518,
3972 ["latail"] = 10521,
3973 ["ratail"] = 10522,
3974 ["lAtail"] = 10523,
3975 ["rAtail"] = 10524,
3976 ["larrfs"] = 10525,
3977 ["rarrfs"] = 10526,
3978 ["larrbfs"] = 10527,
3979 ["rarrbfs"] = 10528,
3980 ["nwarhk"] = 10531,
3981 ["nearhk"] = 10532,
3982 ["searhk"] = 10533,
3983 ["hksearow"] = 10533,
3984 ["swarhk"] = 10534,
3985 ["hkswarow"] = 10534,
3986 ["nwnear"] = 10535,
3987 ["nesear"] = 10536,
3988 ["toea"] = 10536,
3989 ["seswar"] = 10537,
3990 ["tosa"] = 10537,
3991 ["swnwar"] = 10538,
3992 ["rarrc"] = 10547,
3993 ["cudarr"] = 10549,
3994 ["ldca"] = 10550,
3995 ["rdca"] = 10551,
3996 ["cudarrl"] = 10552,
3997 ["larrpl"] = 10553,
3998 ["curarrm"] = 10556,

```

```

3999 ["cularrp"] = 10557,
4000 ["rarrpl"] = 10565,
4001 ["harrcir"] = 10568,
4002 ["Uarrocir"] = 10569,
4003 ["lurdshar"] = 10570,
4004 ["ldrushar"] = 10571,
4005 ["LeftRightVector"] = 10574,
4006 ["RightUpDownVector"] = 10575,
4007 ["DownLeftRightVector"] = 10576,
4008 ["LeftUpDownVector"] = 10577,
4009 ["LeftVectorBar"] = 10578,
4010 ["RightVectorBar"] = 10579,
4011 ["RightUpVectorBar"] = 10580,
4012 ["RightDownVectorBar"] = 10581,
4013 ["DownLeftVectorBar"] = 10582,
4014 ["DownRightVectorBar"] = 10583,
4015 ["LeftUpVectorBar"] = 10584,
4016 ["LeftDownVectorBar"] = 10585,
4017 ["LeftTeeVector"] = 10586,
4018 ["RightTeeVector"] = 10587,
4019 ["RightUpTeeVector"] = 10588,
4020 ["RightDownTeeVector"] = 10589,
4021 ["DownLeftTeeVector"] = 10590,
4022 ["DownRightTeeVector"] = 10591,
4023 ["LeftUpTeeVector"] = 10592,
4024 ["LeftDownTeeVector"] = 10593,
4025 ["lHar"] = 10594,
4026 ["uHar"] = 10595,
4027 ["rHar"] = 10596,
4028 ["dHar"] = 10597,
4029 ["luruhar"] = 10598,
4030 ["ldrdhar"] = 10599,
4031 ["ruluhar"] = 10600,
4032 ["rdldhar"] = 10601,
4033 ["lharul"] = 10602,
4034 ["llhard"] = 10603,
4035 ["rharul"] = 10604,
4036 ["lrhard"] = 10605,
4037 ["udhar"] = 10606,
4038 ["UpEquilibrium"] = 10606,
4039 ["duhar"] = 10607,
4040 ["ReverseUpEquilibrium"] = 10607,
4041 ["RoundImplies"] = 10608,
4042 ["erarr"] = 10609,
4043 ["simrarr"] = 10610,
4044 ["larrsim"] = 10611,
4045 ["rarrsim"] = 10612,

```

```

4046 ["rarrap"] = 10613,
4047 ["ltlarr"] = 10614,
4048 ["gtrarr"] = 10616,
4049 ["subrarr"] = 10617,
4050 ["suplarr"] = 10619,
4051 ["lfisht"] = 10620,
4052 ["rfisht"] = 10621,
4053 ["ufisht"] = 10622,
4054 ["dfisht"] = 10623,
4055 ["lopar"] = 10629,
4056 ["ropar"] = 10630,
4057 ["lbrke"] = 10635,
4058 ["rbrke"] = 10636,
4059 ["lbrkslu"] = 10637,
4060 ["rbrksld"] = 10638,
4061 ["lbrksld"] = 10639,
4062 ["rbrkslu"] = 10640,
4063 ["langd"] = 10641,
4064 ["rangd"] = 10642,
4065 ["lparlt"] = 10643,
4066 ["rpargt"] = 10644,
4067 ["gtlPar"] = 10645,
4068 ["ltrPar"] = 10646,
4069 ["vzigzag"] = 10650,
4070 ["vangrt"] = 10652,
4071 ["angrtvbd"] = 10653,
4072 ["ange"] = 10660,
4073 ["range"] = 10661,
4074 ["dwangle"] = 10662,
4075 ["uwangle"] = 10663,
4076 ["angmsdaa"] = 10664,
4077 ["angmsdab"] = 10665,
4078 ["angmsdac"] = 10666,
4079 ["angmsdad"] = 10667,
4080 ["angmsdae"] = 10668,
4081 ["angmsdaf"] = 10669,
4082 ["angmsdag"] = 10670,
4083 ["angmsdah"] = 10671,
4084 ["bemptyv"] = 10672,
4085 ["demptyv"] = 10673,
4086 ["cemptyv"] = 10674,
4087 ["raemptyv"] = 10675,
4088 ["laemptyv"] = 10676,
4089 ["ohbar"] = 10677,
4090 ["omid"] = 10678,
4091 ["opar"] = 10679,
4092 ["operp"] = 10681,

```

```

4093 ["olcross"] = 10683,
4094 ["odsold"] = 10684,
4095 ["olcir"] = 10686,
4096 ["ofcir"] = 10687,
4097 ["olt"] = 10688,
4098 ["ogt"] = 10689,
4099 ["cirscir"] = 10690,
4100 ["cirE"] = 10691,
4101 ["solb"] = 10692,
4102 ["bsolb"] = 10693,
4103 ["boxbox"] = 10697,
4104 ["trish"] = 10701,
4105 ["rtriltri"] = 10702,
4106 ["LeftTriangleBar"] = 10703,
4107 ["RightTriangleBar"] = 10704,
4108 ["race"] = 10714,
4109 ["iinfin"] = 10716,
4110 ["infintie"] = 10717,
4111 ["nvinfin"] = 10718,
4112 ["eparsl"] = 10723,
4113 ["smeparsl"] = 10724,
4114 ["eqvparsl"] = 10725,
4115 ["lozf"] = 10731,
4116 ["blacklozenge"] = 10731,
4117 ["RuleDelayed"] = 10740,
4118 ["dsol"] = 10742,
4119 ["xodot"] = 10752,
4120 ["bigodot"] = 10752,
4121 ["xoplus"] = 10753,
4122 ["bigoplus"] = 10753,
4123 ["xotime"] = 10754,
4124 ["bigotimes"] = 10754,
4125 ["xuplus"] = 10756,
4126 ["biguplus"] = 10756,
4127 ["xsqcup"] = 10758,
4128 ["bigsqcup"] = 10758,
4129 ["qint"] = 10764,
4130 ["iiiint"] = 10764,
4131 ["fpartint"] = 10765,
4132 ["cirfnint"] = 10768,
4133 ["awint"] = 10769,
4134 ["rppointint"] = 10770,
4135 ["scpointint"] = 10771,
4136 ["npointint"] = 10772,
4137 ["pointint"] = 10773,
4138 ["quatint"] = 10774,
4139 ["intlarhk"] = 10775,

```

```

4140 ["pluscir"] = 10786,
4141 ["plusacir"] = 10787,
4142 ["simplus"] = 10788,
4143 ["plusdu"] = 10789,
4144 ["plussim"] = 10790,
4145 ["plustwo"] = 10791,
4146 ["mcomma"] = 10793,
4147 ["minusdu"] = 10794,
4148 ["loplus"] = 10797,
4149 ["roplus"] = 10798,
4150 ["Cross"] = 10799,
4151 ["timesd"] = 10800,
4152 ["timesbar"] = 10801,
4153 ["smashp"] = 10803,
4154 ["lotimes"] = 10804,
4155 ["rotimes"] = 10805,
4156 ["otimesas"] = 10806,
4157 ["Otimes"] = 10807,
4158 ["odiv"] = 10808,
4159 ["triplus"] = 10809,
4160 ["triminus"] = 10810,
4161 ["tritime"] = 10811,
4162 ["iprod"] = 10812,
4163 ["intprod"] = 10812,
4164 ["amalg"] = 10815,
4165 ["capdot"] = 10816,
4166 ["ncup"] = 10818,
4167 ["ncap"] = 10819,
4168 ["capand"] = 10820,
4169 ["cupor"] = 10821,
4170 ["cupcap"] = 10822,
4171 ["capcup"] = 10823,
4172 ["cupbrcap"] = 10824,
4173 ["capbrcup"] = 10825,
4174 ["cupcup"] = 10826,
4175 ["capcap"] = 10827,
4176 ["ccups"] = 10828,
4177 ["ccaps"] = 10829,
4178 ["ccupssm"] = 10832,
4179 ["And"] = 10835,
4180 ["Or"] = 10836,
4181 ["andand"] = 10837,
4182 ["oror"] = 10838,
4183 ["orslope"] = 10839,
4184 ["andslope"] = 10840,
4185 ["andv"] = 10842,
4186 ["orv"] = 10843,

```

```

4187 ["andd"] = 10844,
4188 ["ord"] = 10845,
4189 ["wedbar"] = 10847,
4190 ["sdote"] = 10854,
4191 ["simdot"] = 10858,
4192 ["congdots"] = 10861,
4193 ["easter"] = 10862,
4194 ["apacir"] = 10863,
4195 ["apE"] = 10864,
4196 ["eplus"] = 10865,
4197 ["pluse"] = 10866,
4198 ["Esim"] = 10867,
4199 ["Colone"] = 10868,
4200 ["Equal"] = 10869,
4201 ["eDDot"] = 10871,
4202 ["ddotseq"] = 10871,
4203 ["equivDD"] = 10872,
4204 ["ltcir"] = 10873,
4205 ["gtcir"] = 10874,
4206 ["ltquest"] = 10875,
4207 ["gtquest"] = 10876,
4208 ["les"] = 10877,
4209 ["LessSlantEqual"] = 10877,
4210 ["leqslant"] = 10877,
4211 ["ges"] = 10878,
4212 ["GreaterSlantEqual"] = 10878,
4213 ["geqslant"] = 10878,
4214 ["lesdot"] = 10879,
4215 ["gesdot"] = 10880,
4216 ["lesdoto"] = 10881,
4217 ["gesdoto"] = 10882,
4218 ["lesdotor"] = 10883,
4219 ["gesdotol"] = 10884,
4220 ["lap"] = 10885,
4221 ["lessapprox"] = 10885,
4222 ["gap"] = 10886,
4223 ["gtrapprox"] = 10886,
4224 ["lne"] = 10887,
4225 ["lneq"] = 10887,
4226 ["gne"] = 10888,
4227 ["gneq"] = 10888,
4228 ["lnap"] = 10889,
4229 ["lnapprox"] = 10889,
4230 ["gnap"] = 10890,
4231 ["gnapprox"] = 10890,
4232 ["lEg"] = 10891,
4233 ["lesseqgtr"] = 10891,

```

```

4234 ["gEl"] = 10892,
4235 ["gtreqqless"] = 10892,
4236 ["lsime"] = 10893,
4237 ["gsime"] = 10894,
4238 ["lsimg"] = 10895,
4239 ["gsiml"] = 10896,
4240 ["lgE"] = 10897,
4241 ["glE"] = 10898,
4242 ["lesges"] = 10899,
4243 ["gesles"] = 10900,
4244 ["els"] = 10901,
4245 ["eqslantless"] = 10901,
4246 ["egs"] = 10902,
4247 ["eqslantgtr"] = 10902,
4248 ["elsdot"] = 10903,
4249 ["egsdot"] = 10904,
4250 ["el"] = 10905,
4251 ["eg"] = 10906,
4252 ["siml"] = 10909,
4253 ["simg"] = 10910,
4254 ["simlE"] = 10911,
4255 ["simgE"] = 10912,
4256 ["LessLess"] = 10913,
4257 ["GreaterGreater"] = 10914,
4258 ["glj"] = 10916,
4259 ["gla"] = 10917,
4260 ["ltcc"] = 10918,
4261 ["gtcc"] = 10919,
4262 ["lescc"] = 10920,
4263 ["gescc"] = 10921,
4264 ["smt"] = 10922,
4265 ["lat"] = 10923,
4266 ["smte"] = 10924,
4267 ["late"] = 10925,
4268 ["bumpE"] = 10926,
4269 ["pre"] = 10927,
4270 ["preceq"] = 10927,
4271 ["PrecedesEqual"] = 10927,
4272 ["sce"] = 10928,
4273 ["succeq"] = 10928,
4274 ["SucceedsEqual"] = 10928,
4275 ["prE"] = 10931,
4276 ["scE"] = 10932,
4277 ["prnE"] = 10933,
4278 ["precneqq"] = 10933,
4279 ["scnE"] = 10934,
4280 ["succneqq"] = 10934,

```



```

4281 ["prap"] = 10935,
4282 ["precapprox"] = 10935,
4283 ["scap"] = 10936,
4284 ["succapprox"] = 10936,
4285 ["prnap"] = 10937,
4286 ["precnapprox"] = 10937,
4287 ["scnap"] = 10938,
4288 ["succnapprox"] = 10938,
4289 ["Pr"] = 10939,
4290 ["Sc"] = 10940,
4291 ["subdot"] = 10941,
4292 ["supdot"] = 10942,
4293 ["subplus"] = 10943,
4294 ["supplus"] = 10944,
4295 ["submult"] = 10945,
4296 ["supmult"] = 10946,
4297 ["subedot"] = 10947,
4298 ["supedot"] = 10948,
4299 ["subE"] = 10949,
4300 ["subseteqq"] = 10949,
4301 ["supE"] = 10950,
4302 ["supseteqq"] = 10950,
4303 ["subsim"] = 10951,
4304 ["supsim"] = 10952,
4305 ["subnE"] = 10955,
4306 ["subsetneqq"] = 10955,
4307 ["supnE"] = 10956,
4308 ["supsetneqq"] = 10956,
4309 ["csub"] = 10959,
4310 ["csup"] = 10960,
4311 ["csube"] = 10961,
4312 ["csupe"] = 10962,
4313 ["subsup"] = 10963,
4314 ["supsub"] = 10964,
4315 ["subsub"] = 10965,
4316 ["supsup"] = 10966,
4317 ["suphsub"] = 10967,
4318 ["supdsub"] = 10968,
4319 ["forkv"] = 10969,
4320 ["topfork"] = 10970,
4321 ["mlcp"] = 10971,
4322 ["Dashv"] = 10980,
4323 ["DoubleLeftTee"] = 10980,
4324 ["Vdashl"] = 10982,
4325 ["Barv"] = 10983,
4326 ["vBar"] = 10984,
4327 ["vBarv"] = 10985,

```

```

4328 ["Vbar"] = 10987,
4329 ["Not"] = 10988,
4330 ["bNot"] = 10989,
4331 ["rnmid"] = 10990,
4332 ["cirmid"] = 10991,
4333 ["midcir"] = 10992,
4334 ["topcir"] = 10993,
4335 ["nhpar"] = 10994,
4336 ["parsim"] = 10995,
4337 ["parsl"] = 11005,
4338 ["fflig"] = 64256,
4339 ["filig"] = 64257,
4340 ["fllig"] = 64258,
4341 ["ffilig"] = 64259,
4342 ["ffllig"] = 64260,
4343 ["Ascr"] = 119964,
4344 ["Cscr"] = 119966,
4345 ["Dscr"] = 119967,
4346 ["Gscr"] = 119970,
4347 ["Jscr"] = 119973,
4348 ["Kscr"] = 119974,
4349 ["Nscr"] = 119977,
4350 ["Oscr"] = 119978,
4351 ["Pscr"] = 119979,
4352 ["Qscr"] = 119980,
4353 ["Sscr"] = 119982,
4354 ["Tscr"] = 119983,
4355 ["Uscr"] = 119984,
4356 ["Vscr"] = 119985,
4357 ["Wscr"] = 119986,
4358 ["Xscr"] = 119987,
4359 ["Yscr"] = 119988,
4360 ["Zscr"] = 119989,
4361 ["ascr"] = 119990,
4362 ["bscr"] = 119991,
4363 ["cscr"] = 119992,
4364 ["dscr"] = 119993,
4365 ["fscr"] = 119995,
4366 ["hscr"] = 119997,
4367 ["iscr"] = 119998,
4368 ["jscr"] = 119999,
4369 ["kscr"] = 120000,
4370 ["lscr"] = 120001,
4371 ["mscr"] = 120002,
4372 ["nscr"] = 120003,
4373 ["pscr"] = 120005,
4374 ["qscr"] = 120006,

```

```

4375 ["rscr"] = 120007,
4376 ["sscr"] = 120008,
4377 ["tscr"] = 120009,
4378 ["uscr"] = 120010,
4379 ["vscr"] = 120011,
4380 ["wscr"] = 120012,
4381 ["xscr"] = 120013,
4382 ["yscr"] = 120014,
4383 ["zscr"] = 120015,
4384 ["Afr"] = 120068,
4385 ["Bfr"] = 120069,
4386 ["Dfr"] = 120071,
4387 ["Efr"] = 120072,
4388 ["Ffr"] = 120073,
4389 ["Gfr"] = 120074,
4390 ["Jfr"] = 120077,
4391 ["Kfr"] = 120078,
4392 ["Lfr"] = 120079,
4393 ["Mfr"] = 120080,
4394 ["Nfr"] = 120081,
4395 ["Ofr"] = 120082,
4396 ["Pfr"] = 120083,
4397 ["Qfr"] = 120084,
4398 ["Sfr"] = 120086,
4399 ["Tfr"] = 120087,
4400 ["Ufr"] = 120088,
4401 ["Vfr"] = 120089,
4402 ["Wfr"] = 120090,
4403 ["Xfr"] = 120091,
4404 ["Yfr"] = 120092,
4405 ["afr"] = 120094,
4406 ["bfr"] = 120095,
4407 ["cfr"] = 120096,
4408 ["dfr"] = 120097,
4409 ["efr"] = 120098,
4410 ["ffr"] = 120099,
4411 ["gfr"] = 120100,
4412 ["hfr"] = 120101,
4413 ["ifr"] = 120102,
4414 ["jfr"] = 120103,
4415 ["kfr"] = 120104,
4416 ["lfr"] = 120105,
4417 ["mfr"] = 120106,
4418 ["nfr"] = 120107,
4419 ["ofr"] = 120108,
4420 ["pfr"] = 120109,
4421 ["qfr"] = 120110,

```

```

4422 ["rfr"] = 120111,
4423 ["sfr"] = 120112,
4424 ["tfr"] = 120113,
4425 ["ufr"] = 120114,
4426 ["vfr"] = 120115,
4427 ["wfr"] = 120116,
4428 ["xfr"] = 120117,
4429 ["yfr"] = 120118,
4430 ["zfr"] = 120119,
4431 ["Aopf"] = 120120,
4432 ["Bopf"] = 120121,
4433 ["Dopf"] = 120123,
4434 ["Eopf"] = 120124,
4435 ["Fopf"] = 120125,
4436 ["Gopf"] = 120126,
4437 ["Iopf"] = 120128,
4438 ["Jopf"] = 120129,
4439 ["Kopf"] = 120130,
4440 ["Lopf"] = 120131,
4441 ["Mopf"] = 120132,
4442 ["Oopf"] = 120134,
4443 ["Sopf"] = 120138,
4444 ["Topf"] = 120139,
4445 ["Uopf"] = 120140,
4446 ["Vopf"] = 120141,
4447 ["Wopf"] = 120142,
4448 ["Xopf"] = 120143,
4449 ["Yopf"] = 120144,
4450 ["aopf"] = 120146,
4451 ["bopf"] = 120147,
4452 ["copf"] = 120148,
4453 ["dopf"] = 120149,
4454 ["eopf"] = 120150,
4455 ["fopf"] = 120151,
4456 ["gopf"] = 120152,
4457 ["hopf"] = 120153,
4458 ["iopf"] = 120154,
4459 ["jopf"] = 120155,
4460 ["kopf"] = 120156,
4461 ["lopf"] = 120157,
4462 ["mopf"] = 120158,
4463 ["nopf"] = 120159,
4464 ["oopf"] = 120160,
4465 ["popf"] = 120161,
4466 ["qopf"] = 120162,
4467 ["ropf"] = 120163,
4468 ["sopf"] = 120164,

```

```

4469 ["topf"] = 120165,
4470 ["uopf"] = 120166,
4471 ["vopf"] = 120167,
4472 ["wopf"] = 120168,
4473 ["xopf"] = 120169,
4474 ["yopf"] = 120170,
4475 ["zopf"] = 120171,
4476 }

```

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

```

4477 function entities.dec_entity(s)
4478 return unicode.utf8.char(tonumber(s))
4479 end

```

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

```

4480 function entities.hex_entity(s)
4481 return unicode.utf8.char(tonumber("0x"..s))
4482 end

```

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

```

4483 function entities.char_entity(s)
4484 local n = character_entities[s]
4485 if n == nil then
4486 return "&" .. s .. ";"
4487 end
4488 return unicode.utf8.char(n)
4489 end

```

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark 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.

```

4490 M.writer = {}

```

The `writer.new` method creates and returns a new T<sub>E</sub>X writer object associated with the Lua interface options (see Section 2.1.3) `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`. All member variables are immutable unless explicitly stated otherwise.

```
4491 function M.writer.new(options)
4492 local self = {}
```

Make `options` available as `writer->options`, so that it is accessible from extensions.

```
4493 self.options = options
```

Parse the `slice` option and define `writer->slice_begin`, `writer->slice_end`, and `writer->is_writing`. The `writer->is_writing` member variable is mutable.

```
4494 local slice_specifiers = {}
4495 for specifier in options.slice:gmatch("[^%s]+") do
4496 table.insert(slice_specifiers, specifier)
4497 end
4498
4499 if #slice_specifiers == 2 then
4500 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4501 local slice_begin_type = self.slice_begin:sub(1, 1)
4502 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4503 self.slice_begin = "^" .. self.slice_begin
4504 end
4505 local slice_end_type = self.slice_end:sub(1, 1)
4506 if slice_end_type ~= "^" and slice_end_type ~= "$" then
4507 self.slice_end = "$" .. self.slice_end
4508 end
4509 elseif #slice_specifiers == 1 then
4510 self.slice_begin = "^" .. slice_specifiers[1]
4511 self.slice_end = "$" .. slice_specifiers[1]
4512 end
4513
4514 if self.slice_begin == "^" and self.slice_end ~= "^" then
4515 self.is_writing = true
4516 else
4517 self.is_writing = false
4518 end
```

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

```
4519 self.suffix = ".tex"
```

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

```
4520 self.space = " "
```

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

```
4521 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
4522 function self.plain(s)
4523 return s
4524 end
```

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

```
4525 function self.paragraph(s)
4526 if not self.is_writing then return "" end
4527 return s
4528 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.

```
4529 function self.pack(name)
4530 return [[\input]] .. name .. [[\relax]]
4531 end
```

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

```
4532 function self.interblocksep()
4533 if not self.is_writing then return "" end
4534 return "\\markdownRendererInterblockSeparator\n{}"
4535 end
```

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

```
4536 self.linebreak = "\\markdownRendererLineBreak\n{}"
```

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

```
4537 self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
4538 function self.hrule()
4539 if not self.is_writing then return "" end
4540 return "\\markdownRendererHorizontalRule{}"
4541 end
```

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```
4542 self.escaped_uri_chars = {
4543 ["{"] = "\\markdownRendererLeftBrace{}",
4544 ["}"] = "\\markdownRendererRightBrace{}",
4545 ["%"] = "\\markdownRendererPercentSign{}",
4546 ["\\"] = "\\markdownRendererBackslash{}",
4547 }
4548 self.escaped_minimal_strings = {
4549 ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4550 ["☒"] = "\\markdownRendererTickedBox{}",

```

```

4551 ["☐"] = "\\markdownRendererHalfTickedBox{}",
4552 ["□"] = "\\markdownRendererUntickedBox{}",
4553 }

```

Define a table `writer->escaped_chars` containing the mapping from special plain  $\text{\TeX}$  characters (including the active pipe character (`|`) of  $\text{\ConTeXt}$ ) that need to be escaped for typeset content.

```

4554 self.escaped_chars = {
4555 ["{"] = "\\markdownRendererLeftBrace{}",
4556 ["}"] = "\\markdownRendererRightBrace{}",
4557 ["%"] = "\\markdownRendererPercentSign{}",
4558 ["\\"] = "\\markdownRendererBackslash{}",
4559 ["#"] = "\\markdownRendererHash{}",
4560 ["$"] = "\\markdownRendererDollarSign{}",
4561 ["&"] = "\\markdownRendererAmpersand{}",
4562 ["_"] = "\\markdownRendererUnderscore{}",
4563 ["^"] = "\\markdownRendererCircumflex{}",
4564 ["~"] = "\\markdownRendererTilde{}",
4565 ["|"] = "\\markdownRendererPipe{}",
4566 }

```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal` tables to create the `writer->escape`, `writer->escape_uri`, and `writer->escape_minimal` escaper functions.

```

4567 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4568 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4569 self.escape_minimal = util.escaper({}, self.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 enabled, use the `writer->escape_minimal`. Otherwise, use the `writer->escape`, and `writer->escape_uri` functions.

```

4570 if options.hybrid then
4571 self.string = self.escape_minimal
4572 self.uri = self.escape_minimal
4573 else
4574 self.string = self.escape
4575 self.uri = self.escape_uri
4576 end

```

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

```

4577 function self.code(s)
4578 return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
4579 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.

```
4580 function self.link(lab,src,tit)
4581 return {"\\markdownRendererLink{" ,lab,"} ",
4582 "{" ,self.escape(src),"} ",
4583 "{" ,self.uri(src),"} ",
4584 "{" ,self.string(tit or ""),"} "}
4585 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.

```
4586 function self.image(lab,src,tit)
4587 return {"\\markdownRendererImage{" ,lab,"} ",
4588 "{" ,self.string(src),"} ",
4589 "{" ,self.uri(src),"} ",
4590 "{" ,self.string(tit or ""),"} "}
4591 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.

```
4592 function self.bulletlist(items,tight)
4593 if not self.is_writing then return "" end
4594 local buffer = {}
4595 for _,item in ipairs(items) do
4596 buffer[#buffer + 1] = self.bulletitem(item)
4597 end
4598 local contents = util.intersperse(buffer,"\n")
4599 if tight and options.tightLists then
4600 return {"\\markdownRendererULBeginTight\n",contents,
4601 "\n\\markdownRendererULEndTight "}
4602 else
4603 return {"\\markdownRendererULBegin\n",contents,
4604 "\n\\markdownRendererULEnd "}
4605 end
4606 end
```

Define `writer->bulletitem` as a function that will transform an input bulleted list item to the output format, where `s` is the text of the list item.

```
4607 function self.bulletitem(s)
4608 return {"\\markdownRendererULItem ",s,
4609 "\\markdownRendererULItemEnd "}
4610 end
```

Define `writer->orderedlist` 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 is the number of the first list item.

```

4611 function self.orderedlist(items,tight,startnum)
4612 if not self.is_writing then return "" end
4613 local buffer = {}
4614 local num = startnum
4615 for _,item in ipairs(items) do
4616 buffer[#buffer + 1] = self.ordereditem(item,num)
4617 if num ~= nil then
4618 num = num + 1
4619 end
4620 end
4621 local contents = util.intersperse(buffer,"\n")
4622 if tight and options.tightLists then
4623 return {"\\markdownRendererOlBeginTight\n",contents,
4624 "\n\\markdownRendererOlEndTight "}
4625 else
4626 return {"\\markdownRendererOlBegin\n",contents,
4627 "\n\\markdownRendererOlEnd "}
4628 end
4629 end

```

Define `writer->ordereditem` as a function that will transform an input ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

4630 function self.ordereditem(s,num)
4631 if num ~= nil then
4632 return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
4633 "\\markdownRendererOlItemEnd "}
4634 else
4635 return {"\\markdownRendererOlItem ",s,
4636 "\\markdownRendererOlItemEnd "}
4637 end
4638 end

```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

4639 function self.inline_html_comment(contents)
4640 return {"\\markdownRendererInlineHtmlComment{",contents,""}
4641 end

```

Define `writer->block_html_comment` as a function that will transform the contents of a block HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

4642 function self.block_html_comment(contents)
4643 if not self.is_writing then return "" end

```

```

4644 return {"\\markdownRendererBlockHtmlCommentBegin\\n",contents,
4645 "\\n\\markdownRendererBlockHtmlCommentEnd "}
4646 end

```

Define `writer->inline_html_tag` as a function that will transform the contents of an opening, closing, or empty inline HTML tag to the output format, where `contents` are the contents of the HTML tag.

```

4647 function self.inline_html_tag(contents)
4648 return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
4649 end

```

Define `writer->block_html_element` as a function that will transform the contents of a block HTML element to the output format, where `s` are the contents of the HTML element.

```

4650 function self.block_html_element(s)
4651 if not self.is_writing then return "" end
4652 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4653 return {"\\markdownRendererInputBlockHtmlElement{" ,name,""}"}
4654 end

```

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

```

4655 function self.emphasis(s)
4656 return {"\\markdownRendererEmphasis{" ,s,""}"}
4657 end

```

Define `writer->checkbox` as a function that will transform a number `f` to the output format.

```

4658 function self.checkbox(f)
4659 if f == 1.0 then
4660 return "☑ "
4661 elseif f == 0.0 then
4662 return "☐ "
4663 else
4664 return "☐ "
4665 end
4666 end

```

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

```

4667 function self.strong(s)
4668 return {"\\markdownRendererStrongEmphasis{" ,s,""}"}
4669 end

```

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

```

4670 function self.blockquote(s)
4671 if #util.rope_to_string(s) == 0 then return "" end

```

```

4672 return {"\\markdownRendererBlockQuoteBegin\\n",s,
4673 "\\n\\markdownRendererBlockQuoteEnd "}
4674 end

```

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

```

4675 function self.verbatim(s)
4676 if not self.is_writing then return "" end
4677 s = string.gsub(s, '[\\r\\n%s]*$', '')
4678 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4679 return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4680 end

```

Define `writer->document` as a function that will transform a document `d` to the output format.

```

4681 function self.document(d)
4682 local active_attributes = self.active_attributes
4683 local buf = {"\\markdownRendererDocumentBegin\\n", d}
4684
4685 -- pop attributes for sections that have ended
4686 if options.headerAttributes and self.is_writing then
4687 while #active_attributes > 0 do
4688 local attributes = active_attributes[#active_attributes]
4689 if #attributes > 0 then
4690 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4691 end
4692 table.remove(active_attributes, #active_attributes)
4693 end
4694 end
4695
4696 table.insert(buf, "\\markdownRendererDocumentEnd")
4697
4698 return buf
4699 end

```

Define `writer->active_attributes` as a stack of attributes of the headings that are currently active. The `writer->active_headings` member variable is mutable.

```

4700 self.active_attributes = {}

```

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

```

4701 function self.heading(s, level, attributes)
4702 attributes = attributes or {}
4703 for i = 1, #attributes do
4704 attributes[attributes[i]] = true
4705 end
4706
4707 local active_attributes = self.active_attributes

```

```

4708 local slice_begin_type = self.slice_begin:sub(1, 1)
4709 local slice_begin_identifier = self.slice_begin:sub(2) or ""
4710 local slice_end_type = self.slice_end:sub(1, 1)
4711 local slice_end_identifier = self.slice_end:sub(2) or ""
4712
4713 local buf = {}
4714
4715 -- push empty attributes for implied sections
4716 while #active_attributes < level-1 do
4717 table.insert(active_attributes, {})
4718 end
4719
4720 -- pop attributes for sections that have ended
4721 while #active_attributes >= level do
4722 local active_identifiers = active_attributes[#active_attributes]
4723 -- tear down all active attributes at slice end
4724 if active_identifiers["#" .. slice_end_identifier] ~= nil
4725 and slice_end_type == "$" then
4726 for header_level = #active_attributes, 1, -1 do
4727 if options.headerAttributes and #active_attributes[header_level] > 0 then
4728 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4729 end
4730 end
4731 self.is_writing = false
4732 end
4733 table.remove(active_attributes, #active_attributes)
4734 if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4735 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4736 end
4737 -- apply all active attributes at slice beginning
4738 if active_identifiers["#" .. slice_begin_identifier] ~= nil
4739 and slice_begin_type == "$" then
4740 for header_level = 1, #active_attributes do
4741 if options.headerAttributes and #active_attributes[header_level] > 0 then
4742 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4743 end
4744 end
4745 self.is_writing = true
4746 end
4747 end
4748
4749 -- tear down all active attributes at slice end
4750 if attributes["#" .. slice_end_identifier] ~= nil
4751 and slice_end_type == "^" then
4752 for header_level = #active_attributes, 1, -1 do
4753 if options.headerAttributes and #active_attributes[header_level] > 0 then
4754 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")

```

```

4755 end
4756 end
4757 self.is_writing = false
4758 end
4759
4760 -- push attributes for the new section
4761 table.insert(active_attributes, attributes)
4762 if self.is_writing and options.headerAttributes and #attributes > 0 then
4763 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4764 end
4765
4766 -- apply all active attributes at slice beginning
4767 if attributes["#" .. slice_begin_idenfifier] ~= nil
4768 and slice_begin_type == "^" then
4769 for header_level = 1, #active_attributes do
4770 if options.headerAttributes and #active_attributes[header_level] > 0 then
4771 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4772 end
4773 end
4774 self.is_writing = true
4775 end
4776
4777 if self.is_writing then
4778 table.sort(attributes)
4779 local key, value
4780 for i = 1, #attributes do
4781 if attributes[i]:sub(1, 1) == "#" then
4782 table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
4783 attributes[i]:sub(2), "}"})
4784 elseif attributes[i]:sub(1, 1) == "." then
4785 table.insert(buf, {"\\markdownRendererAttributeClassName{",
4786 attributes[i]:sub(2), "}"})
4787 else
4788 key, value = attributes[i]:match("([^\s=]+)%s*=%s*(.*)")
4789 table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
4790 key, "{", value, "}"})
4791 end
4792 end
4793 end
4794
4795 local cmd
4796 level = level + options.shiftHeadings
4797 if level <= 1 then
4798 cmd = "\\markdownRendererHeadingOne"
4799 elseif level == 2 then
4800 cmd = "\\markdownRendererHeadingTwo"
4801 elseif level == 3 then

```

```

4802 cmd = "\\markdownRendererHeadingThree"
4803 elseif level == 4 then
4804 cmd = "\\markdownRendererHeadingFour"
4805 elseif level == 5 then
4806 cmd = "\\markdownRendererHeadingFive"
4807 elseif level >= 6 then
4808 cmd = "\\markdownRendererHeadingSix"
4809 else
4810 cmd = ""
4811 end
4812 if self.is_writing then
4813 table.insert(buf, {cmd, "{", s, "}"})
4814 end
4815
4816 return buf
4817 end

```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```

4818 function self.get_state()
4819 return {
4820 is_writing=self.is_writing,
4821 active_attributes={table.unpack(self.active_attributes)},
4822 }
4823 end

```

Define `writer->set_state` as a function that restores the input state `s` and returns the previous state of the writer.

```

4824 function self.set_state(s)
4825 local previous_state = self.get_state()
4826 for key, value in pairs(s) do
4827 self[key] = value
4828 end
4829 return previous_state
4830 end

```

Define `writer->defer_call` as a function that will encapsulate the input function `f`, so that `f` is called with the state of the writer at the time of calling `writer->defer_call`.

```

4831 function self.defer_call(f)
4832 local previous_state = self.get_state()
4833 return function(...)
4834 local state = self.set_state(previous_state)
4835 local return_value = f(...)
4836 self.set_state(state)
4837 return return_value
4838 end
4839 end

```

```

4840
4841 return self
4842 end

```

### 3.1.4 Parsers

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

```

4843 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

4844 parsers.percent = P("%")
4845 parsers.at = P("@")
4846 parsers.comma = P(",")
4847 parsers.asterisk = P("*")
4848 parsers.dash = P("-")
4849 parsers.plus = P("+")
4850 parsers.underscore = P("_")
4851 parsers.period = P(".")
4852 parsers.hash = P("#")
4853 parsers.ampersand = P("&")
4854 parsers.backtick = P("`")
4855 parsers.less = P("<")
4856 parsers.more = P(">")
4857 parsers.space = P(" ")
4858 parsers.squote = P("'")
4859 parsers.dquote = P('"')
4860 parsers.lparent = P("(")
4861 parsers.rparent = P(")")
4862 parsers.lbracket = P("[")
4863 parsers.rbracket = P("]")
4864 parsers.lbrace = P("{")
4865 parsers.rbrace = P("}")
4866 parsers.circumflex = P("^")
4867 parsers.slash = P("/")
4868 parsers.equal = P("=")
4869 parsers.colon = P(":")
4870 parsers.semicolon = P(";")
4871 parsers.exclamation = P("!")
4872 parsers.pipe = P("|")
4873 parsers.tilde = P("~")
4874 parsers.backslash = P("\\")
4875 parsers.tab = P("\t")
4876 parsers.newline = P("\n")
4877 parsers.tightblocksep = P("\001")
4878

```



```

4879 parsers.digit = R("09")
4880 parsers.hexdigit = R("09","af","AF")
4881 parsers.letter = R("AZ","az")
4882 parsers.alphanumeric = R("AZ","az","09")
4883 parsers.keyword = parsers.letter
4884 * parsers.alphanumeric^0
4885 parsers.internal_punctuation = S(".;,.?")
4886
4887 parsers.doubleasterisks = P("**")
4888 parsers.doubleunderscores = P("__")
4889 parsers.doubletildes = P("~")
4890 parsers.fourspace = P(" ")
4891
4892 parsers.any = P(1)
4893 parsers.fail = parsers.any - 1
4894
4895 parsers.escapable = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
4896 parsers.anyescaped = parsers.backslash / " " * parsers.escapable
4897 + parsers.any
4898
4899 parsers.spacechar = S("\t ")
4900 parsers.spacing = S(" \n\r\t")
4901 parsers.nonspacechar = parsers.any - parsers.spacing
4902 parsers.optionalspace = parsers.spacechar^0
4903
4904 parsers.normalchar = parsers.any - (V("SpecialChar")
4905 + parsers.spacing
4906 + parsers.tightblocksep)
4907 parsers.eof = -parsers.any
4908 parsers.nonindentpace = parsers.space^-3 * - parsers.spacechar
4909 parsers.indent = parsers.space^-3 * parsers.tab
4910 + parsers.fourspace / " "
4911 parsers.linechar = P(1 - parsers.newline)
4912
4913 parsers.blankline = parsers.optionalspace
4914 * parsers.newline / "\n"
4915 parsers.blanklines = parsers.blankline^0
4916 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0
4917 parsers.indentedline = parsers.indent / " "
4918 * C(parsers.linechar^1 * parsers.newline^-
1)
4919 parsers.optionallyindentedline = parsers.indent^-1 / " "
4920 * C(parsers.linechar^1 * parsers.newline^-
1)
4921 parsers.sp = parsers.spacing^0
4922 parsers.spnl = parsers.optionalspace

```

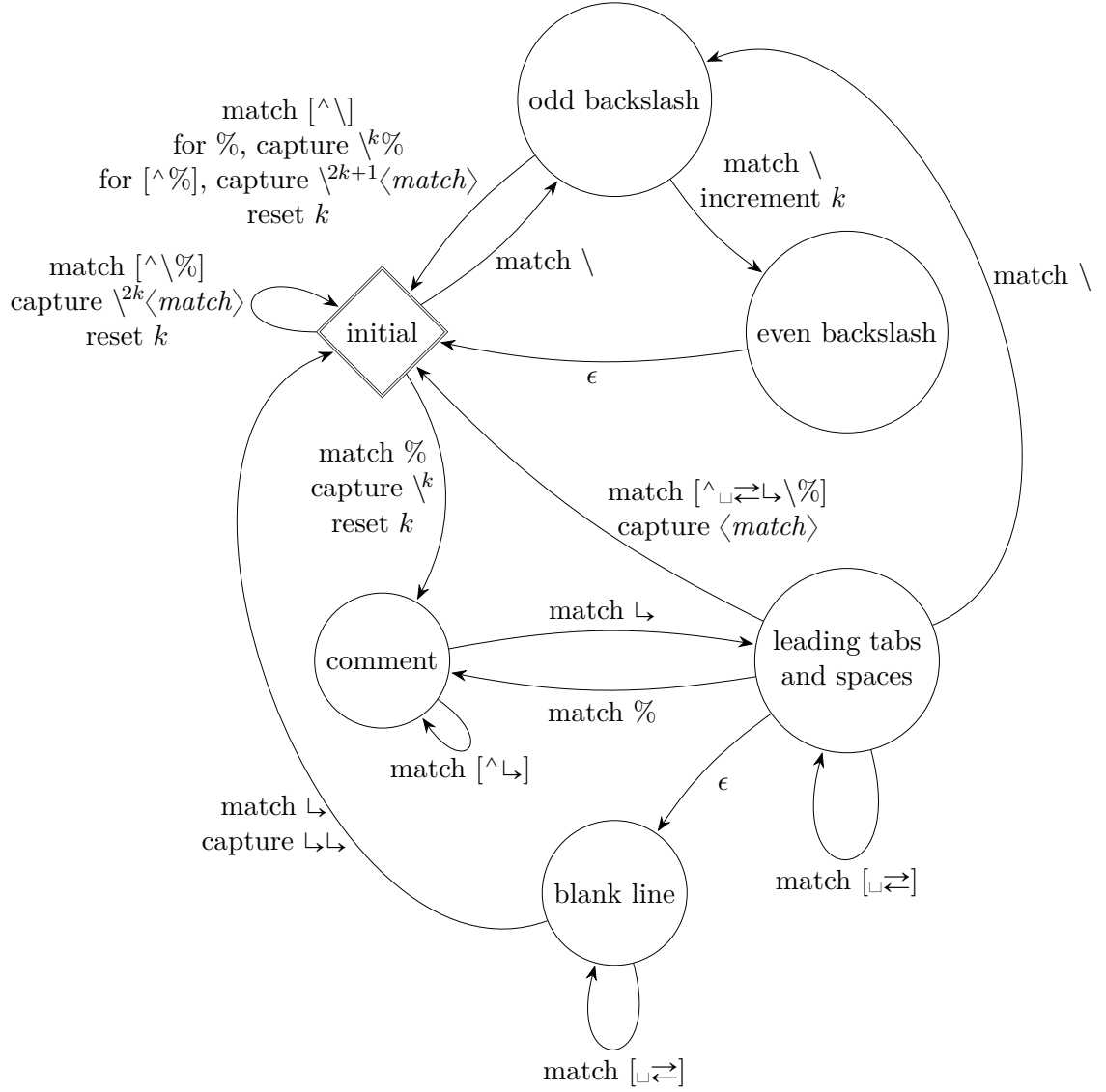
```

4923 * (parsers.newline * parsers.optionalspace)^-
1
4924 parsers.line = parsers.linechar^0 * parsers.newline
4925 parsers.nonemptyline = parsers.line - parsers.blankline

The parsers.commented_line^1 parser recognizes the regular language of TEX
comments, see an equivalent finite automaton in Figure 6.

4926 parsers.commented_line_letter = parsers.linechar
4927 + parsers.newline
4928 - parsers.backslash
4929 - parsers.percent
4930 parsers.commented_line = Cg(Cc(""), "backslashes")
4931 * ((#(parsers.commented_line_letter
4932 - parsers.newline)
4933 * Cb("backslashes")
4934 * Cs(parsers.commented_line_letter
4935 - parsers.newline)^1 -- initial
4936 * Cg(Cc(""), "backslashes"))
4937 + #(parsers.backslash * parsers.backslash)
4938 * Cg((parsers.backslash -- even backslash
4939 * parsers.backslash)^1, "backslashes")
4940 + (parsers.backslash
4941 * (#parsers.percent
4942 * Cb("backslashes")
4943 / function(backslashes)
4944 return string.rep("\\", #backslashes / 2)
4945 end
4946 * C(parsers.percent)
4947 + #parsers.commented_line_letter
4948 * Cb("backslashes")
4949 * Cc("\\")
4950 * C(parsers.commented_line_letter))
4951 * Cg(Cc(""), "backslashes")))^0
4952 * (#parsers.percent
4953 * Cb("backslashes")
4954 / function(backslashes)
4955 return string.rep("\\", #backslashes / 2)
4956 end
4957 * ((parsers.percent -- comment
4958 * parsers.line
4959 * #parsers.blankline) -- blank line
4960 / "\\n"
4961 + parsers.percent -- comment
4962 * parsers.line
4963 * parsers.optionalspace) -- leading tabs and spaces
4964 + #(parsers.newline)
4965 * Cb("backslashes")

```



**Figure 6: A pushdown automaton that recognizes TeX comments**

```

4966 * C(parsers.newline))
4967
4968 parsers.chunk = parsers.line * (parsers.optionallyindentedline
4969 - parsers.blankline)^0
4970
4971 parsers.attribute_key_char = parsers.alphanumeric + S("_-")
4972 parsers.attribute_key = (parsers.attribute_key_char
4973 - parsers.dash - parsers.digit)
4974 * parsers.attribute_key_char^0
4975 parsers.attribute_value = ((parsers.dquote / "\"")
4976 * (parsers.anyescaped - parsers.dquote)^0
4977 * (parsers.dquote / "\""))
4978 + (parsers.anyescaped - parsers.dquote - parsers.rbrace
4979 - parsers.space)^0
4980
4981 -- block followed by 0 or more optionally
4982 -- indented blocks with first line indented.
4983 parsers.indented_blocks = function(bl)
4984 return Cs(bl
4985 * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
4986 * (parsers.blankline^1 + parsers.eof))
4987 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

4988 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
4989
4990 parsers.bullet = (parsers.bulletchar * #parsers.spacing
4991 * (parsers.tab + parsers.space^-3)
4992 + parsers.space * parsers.bulletchar * #parsers.spacing
4993 * (parsers.tab + parsers.space^-2)
4994 + parsers.space * parsers.space * parsers.bulletchar
4995 * #parsers.spacing
4996 * (parsers.tab + parsers.space^-1)
4997 + parsers.space * parsers.space * parsers.space
4998 * parsers.bulletchar * #parsers.spacing
4999)
5000
5001 local function tickbox(interior)
5002 return parsers.optionalspace * parsers.lbracket
5003 * interior * parsers.rbracket * parsers.spacechar^1
5004 end
5005
5006 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5007 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5008 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
5009

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```
5010 parsers.openticks = Cg(parsers.backtick^1, "ticks")
5011
5012 local function captures_equal_length(_,i,a,b)
5013 return #a == #b and i
5014 end
5015
5016 parsers.closeticks = parsers.space^-1
5017 * Cmt(C(parsers.backtick^1)
5018 * Cb("ticks"), captures_equal_length)
5019
5020 parsers.intickschar = (parsers.any - S(" \n\r`"))
5021 + (parsers.newline * -parsers.blankline)
5022 + (parsers.space - parsers.closeticks)
5023 + (parsers.backtick^1 - parsers.closeticks)
5024
5025 parsers.inticks = parsers.openticks * parsers.space^-1
5026 * C(parsers.intickschar^0) * parsers.closeticks
```

### 3.1.4.4 Parsers Used for Fenced Code Blocks

```
5027 local function captures_geq_length(_,i,a,b)
5028 return #a >= #b and i
5029 end
5030
5031 parsers.infostring = (parsers.linechar - (parsers.backtick
5032 + parsers.space^1 * (parsers.newline + parsers.eof)))^0
5033
5034 local fenceindent
5035 parsers.fencehead = function(char)
5036 return C(parsers.nonindentSPACE) / function(s) fenceindent = #s end
5037 * Cg(char^3, "fencelength")
5038 * parsers.optionalspace * C(parsers.infostring)
5039 * parsers.optionalspace * (parsers.newline + parsers.eof)
5040 end
5041
5042 parsers.fencetail = function(char)
5043 return parsers.nonindentSPACE
5044 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
5045 * parsers.optionalspace * (parsers.newline + parsers.eof)
5046 + parsers.eof
5047 end
5048
5049 parsers.fencedline = function(char)
5050 return C(parsers.line - parsers.fencetail(char))
5051 / function(s)
5052 local i = 1
```

```

5053 local remaining = fenceindent
5054 while true do
5055 local c = s:sub(i, i)
5056 if c == " " and remaining > 0 then
5057 remaining = remaining - 1
5058 i = i + 1
5059 elseif c == "\t" and remaining > 3 then
5060 remaining = remaining - 4
5061 i = i + 1
5062 else
5063 break
5064 end
5065 end
5066 return s:sub(i)
5067 end
5068 end

```

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```

5069 parsers.leader = parsers.space^-3
5070
5071 -- content in balanced brackets, parentheses, or quotes:
5072 parsers.bracketed = P{ parsers.lbracket
5073 * ((parsers.backslash / "\"" * parsers.rbracket
5074 + parsers.any - (parsers.lbracket
5075 + parsers.rbracket
5076 + parsers.blankline^2)
5077) + V(1))^0
5078 * parsers.rbracket }
5079
5080 parsers.inparens = P{ parsers.lparent
5081 * ((parsers.anyescaped - (parsers.lparent
5082 + parsers.rparent
5083 + parsers.blankline^2)
5084) + V(1))^0
5085 * parsers.rparent }
5086
5087 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
5088 * ((parsers.anyescaped - (parsers.squote
5089 + parsers.blankline^2)
5090) + V(1))^0
5091 * parsers.squote }
5092
5093 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
5094 * ((parsers.anyescaped - (parsers.dquote
5095 + parsers.blankline^2)
5096) + V(1))^0

```

```

5097 * parsers.dquote }
5098
5099 -- bracketed tag for markdown links, allowing nested brackets:
5100 parsers.tag = parsers.lbracket
5101 * Cs((parsers.alphanumeric^1
5102 + parsers.bracketed
5103 + parsers.inticks
5104 + (parsers.backslash / "\"" * parsers.rbracket
5105 + parsers.any
5106 - (parsers.rbracket + parsers.blankline^2)))^0)
5107 * parsers.rbracket
5108
5109 -- url for markdown links, allowing nested brackets:
5110 parsers.url = parsers.less * Cs((parsers.anyescaped
5111 - parsers.more)^0)
5112 * parsers.more
5113 + Cs((parsers.inparens + (parsers.anyescaped
5114 - parsers.spacing
5115 - parsers.rparent))^1)
5116
5117 -- quoted text, possibly with nested quotes:
5118 parsers.title_s = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5119 + parsers.squoted)^0)
5120 * parsers.squote
5121
5122 parsers.title_d = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5123 + parsers.dquoted)^0)
5124 * parsers.dquote
5125
5126 parsers.title_p = parsers.lparent
5127 * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5128 * parsers.rparent
5129
5130 parsers.title = parsers.title_d + parsers.title_s + parsers.title_p
5131
5132 parsers.optionaltitle
5133 = parsers.spnl * parsers.title * parsers.spacechar^0
5134 + Cc("")

```

### 3.1.4.6 Parsers Used for HTML

```

5135 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5136 parsers.keyword_exact = function(s)
5137 local parser = P(0)
5138 for i=1,#s do
5139 local c = s:sub(i,i)
5140 local m = c .. upper(c)

```

```

5141 parser = parser * S(m)
5142 end
5143 return parser
5144 end
5145
5146 parsers.block_keyword =
5147 parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5148 parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5149 parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5150 parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5151 parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5152 parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5153 parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5154 parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5155 parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5156 parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5157 parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5158 parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5159 parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5160 parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5161 parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5162 parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5163 parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5164 parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5165
5166 -- There is no reason to support bad html, so we expect quoted attributes
5167 parsers.htmlattributevalue
5168 = parsers.quote * (parsers.any - (parsers.blankline
5169 + parsers.quote))^0
5170 * parsers.quote
5171 + parsers.dquote * (parsers.any - (parsers.blankline
5172 + parsers.dquote))^0
5173 * parsers.dquote
5174
5175 parsers.htmlattribute = parsers.spacing^1
5176 * (parsers.alphanumeric + S("_-"))^1
5177 * parsers.sp * parsers.equal * parsers.sp
5178 * parsers.htmlattributevalue
5179
5180 parsers.htmlcomment = P("<!--")
5181 * parsers.optionalspace
5182 * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5183 * parsers.optionalspace
5184 * P("-->")
5185
5186 parsers.htmlinstruction = P("<?") * (parsers.any - P(">"))^0 * P(">")
5187

```



```

5188 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5189 * parsers.sp * parsers.more
5190
5191 parsers.openelt_exact = function(s)
5192 return parsers.less * parsers.sp * parsers.keyword_exact(s)
5193 * parsers.htmlattribute^0 * parsers.sp * parsers.more
5194 end
5195
5196 parsers.openelt_block = parsers.sp * parsers.block_keyword
5197 * parsers.htmlattribute^0 * parsers.sp * parsers.more
5198
5199 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5200 * parsers.keyword * parsers.sp * parsers.more
5201
5202 parsers.closeelt_exact = function(s)
5203 return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5204 * parsers.sp * parsers.more
5205 end
5206
5207 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5208 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5209 * parsers.more
5210
5211 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5212 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5213 * parsers.more
5214
5215 parsers.displaytext = (parsers.any - parsers.less)^1
5216
5217 -- return content between two matched HTML tags
5218 parsers.in_matched = function(s)
5219 return { parsers.openelt_exact(s)
5220 * (V(1) + parsers.displaytext
5221 + (parsers.less - parsers.closeelt_exact(s)))^0
5222 * parsers.closeelt_exact(s) }
5223 end
5224
5225 local function parse_matched_tags(s,pos)
5226 local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5227 return lpeg.match(parsers.in_matched(t),s,pos-1)
5228 end
5229
5230 parsers.in_matched_block_tags = parsers.less
5231 * Cmt(#parsers.openelt_block, parse_matched_tags)
5232

```

#### 3.1.4.7 Parsers Used for HTML Entities

```
5233 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5234 * C(parsers.hexdigit^1) * parsers.semicolon
5235 parsers.decentity = parsers.ampersand * parsers.hash
5236 * C(parsers.digit^1) * parsers.semicolon
5237 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5238 * parsers.semicolon
```

#### 3.1.4.8 Helpers for References

```
5239 -- parse a reference definition: [foo]: /bar "title"
5240 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5241 * parsers.spacechar^0 * parsers.url
5242 * parsers.optionaltitle * parsers.blankline^1
```

#### 3.1.4.9 Inline Elements

```
5243 parsers.Inline = V("Inline")
5244 parsers.IndentedInline = V("IndentedInline")
5245
5246 -- parse many p between starter and ender
5247 parsers.between = function(p, starter, ender)
5248 local ender2 = B(parsers.nonspacechar) * ender
5249 return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5250 end
5251
5252 parsers.urlchar = parsers.anyescaped - parsers.newline - parsers.more
```

#### 3.1.4.10 Block Elements

```
5253 parsers.lineof = function(c)
5254 return (parsers.leader * (P(c) * parsers.optionalspace)^3
5255 * (parsers.newline * parsers.blankline^1
5256 + parsers.newline^-1 * parsers.eof))
5257 end
```

#### 3.1.4.11 Headings

```
5258 parsers.heading_attribute = (parsers.dash * Cc(".unnumbered"))
5259 + C((parsers.hash + parsers.period)
5260 * parsers.attribute_key)
5261 + Cs(parsers.attribute_key
5262 * parsers.optionalspace * parsers.equal * parsers.optional
5263 * parsers.attribute_value)
5264 parsers.HeadingAttributes = parsers.lbrace
5265 * parsers.optionalspace
5266 * parsers.heading_attribute
5267 * (parsers.spacechar^1
```

```

5268 * parsers.heading_attribute)^0
5269 * parsers.optionalspace
5270 * parsers.rbrace
5271
5272 -- parse Atx heading start and return level
5273 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
5274 * -parsers.hash / length
5275
5276 -- parse setext header ending and return level
5277 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5278
5279 local function strip_atx_end(s)
5280 return s:gsub("#%s*\n$", "")
5281 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.

The `reader.new` method creates and returns a new TeX reader object associated with the Lua interface options (see Section 2.1.3) `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`.

```

5282 M.reader = {}
5283 function M.reader.new(writer, options)
5284 local self = {}

```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```

5285 self.writer = writer
5286 self.options = options

```

Create a `reader->parsers` hash table that stores PEG patterns that depend on the received `options`. Make `reader->parsers` inherit from the global `parsers` table.

```

5287 self.parsers = {}
5288 (function(parsers)
5289 setmetatable(self.parsers, {
5290 __index = function (_, key)
5291 return parsers[key]
5292 end
5293 })
5294 end)(parsers)

```

Make `reader->parsers` available as a local `parsers` variable that will shadow the global `parsers` table and will make `reader->parsers` easier to type in the rest of the reader code.

```
5295 local parsers = self.parsers
```

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

```
5296 function self.normalize_tag(tag)
5297 return string.lower(
5298 gsub(util.rope_to_string(tag), "[\n\r\t]+", " "))
5299 end
```

Define `iterlines` as a function that iterates over the lines of the input string `s`, transforms them using an input function `f`, and reassembles them into a new string, which it returns.

```
5300 local function iterlines(s, f)
5301 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5302 return util.rope_to_string(rope)
5303 end
```

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

```
5304 if options.preserveTabs then
5305 self.expandtabs = function(s) return s end
5306 else
5307 self.expandtabs = function(s)
5308 if s:find("\t") then
5309 return iterlines(s, util.expand_tabs_in_line)
5310 else
5311 return s
5312 end
5313 end
5314 end
```

**3.1.5.2 High-Level Parser Functions** Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
5315 self.parser_functions = {}
5316 self.create_parser = function(name, grammar, toplevel)
5317 self.parser_functions[name] = function(str)
```

If the parser function is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```

5318 if toplevel and options.stripIndent then
5319 local min_prefix_length, min_prefix = nil, ''
5320 str = iterlines(str, function(line)
5321 if lpeg.match(parsers.nonemptyline, line) == nil then
5322 return line
5323 end
5324 line = util.expand_tabs_in_line(line)
5325 local prefix = lpeg.match(C(parsers.optionalspace), line)
5326 local prefix_length = #prefix
5327 local is_shorter = min_prefix_length == nil
5328 is_shorter = is_shorter or prefix_length < min_prefix_length
5329 if is_shorter then
5330 min_prefix_length, min_prefix = prefix_length, prefix
5331 end
5332 return line
5333 end)
5334 str = str:gsub('^' .. min_prefix, '')
5335 end

```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain TeX comments from the input string `str` together with the trailing newline characters.

```

5336 if toplevel and (options.texComments or options.hybrid) then
5337 str = lpeg.match(Ct(parsers.commented_line^1), str)
5338 str = util.rope_to_string(str)
5339 end
5340 local res = lpeg.match(grammar(), str)
5341 if res == nil then
5342 error(format("%s failed on:\n%s", name, str:sub(1,20)))
5343 else
5344 return res
5345 end
5346 end
5347 end
5348
5349 self.create_parser("parse_blocks",
5350 function()
5351 return parsers.blocks
5352 end, true)
5353
5354 self.create_parser("parse_blocks_nested",
5355 function()
5356 return parsers.blocks_nested

```

```

5357 end, false)
5358
5359 self.create_parser("parse_inlines",
5360 function()
5361 return parsers.inlines
5362 end, false)
5363
5364 self.create_parser("parse_inlines_no_link",
5365 function()
5366 return parsers.inlines_no_link
5367 end, false)
5368
5369 self.create_parser("parse_inlines_no_inline_note",
5370 function()
5371 return parsers.inlines_no_inline_note
5372 end, false)
5373
5374 self.create_parser("parse_inlines_no_html",
5375 function()
5376 return parsers.inlines_no_html
5377 end, false)
5378
5379 self.create_parser("parse_inlines_nbsp",
5380 function()
5381 return parsers.inlines_nbsp
5382 end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```

5383 if options.hashEnumerators then
5384 parsers.dig = parsers.digit + parsers.hash
5385 else
5386 parsers.dig = parsers.digit
5387 end
5388
5389 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5390 + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5391 * (parsers.tab + parsers.space^1)
5392 + C(parsers.dig * parsers.period) * #parsers.spacing
5393 * (parsers.tab + parsers.space^-2)
5394 + parsers.space * C(parsers.dig^2 * parsers.period)
5395 * #parsers.spacing
5396 + parsers.space * C(parsers.dig * parsers.period)
5397 * #parsers.spacing
5398 * (parsers.tab + parsers.space^-1)
5399 + parsers.space * parsers.space * C(parsers.dig^1
5400 * parsers.period) * #parsers.spacing

```

#### 3.1.5.4 Parsers Used for Blockquotes (local)

```
5401 -- strip off leading > and indents, and run through blocks
5402 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space~
5403 1)/""
5404 * parsers.linechar^0 * parsers.newline)^1
5405 * (-(parsers.leader * parsers.more
5406 + parsers.blankline) * parsers.linechar^1
5407 * parsers.newline)^0
5408 if not options.breakableBlockquotes then
5409 parsers.blockquote_body = parsers.blockquote_body
5410 * (parsers.blankline^0 / "")
5411 end
```

#### 3.1.5.5 Parsers Used for Footnotes (local)

#### 3.1.5.6 Helpers for Links and References (local)

```
5412 -- List of references defined in the document
5413 local references
5414
5415 -- add a reference to the list
5416 local function register_link(tag,url,title)
5417 references[self.normalize_tag(tag)] = { url = url, title = title }
5418 return ""
5419 end
5420
5421 -- lookup link reference and return either
5422 -- the link or nil and fallback text.
5423 local function lookup_reference(label,sps,tag)
5424 local tagpart
5425 if not tag then
5426 tag = label
5427 tagpart = ""
5428 elseif tag == "" then
5429 tag = label
5430 tagpart = "[]"
5431 else
5432 tagpart = {"[",
5433 self.parser_functions.parse_inlines(tag),
5434 "]" }
5435 end
5436 if sps then
5437 tagpart = {sps, tagpart}
5438 end
5439 local r = references[self.normalize_tag(tag)]
5440 if r then
```

```

5441 return r
5442 else
5443 return nil, {"[",
5444 self.parser_functions.parse_inlines(label),
5445 "]", tagpart}
5446 end
5447 end
5448
5449 -- lookup link reference and return a link, if the reference is found,
5450 -- or a bracketed label otherwise.
5451 local function indirect_link(label,sps,tag)
5452 return writer.defer_call(function()
5453 local r,fallback = lookup_reference(label,sps,tag)
5454 if r then
5455 return writer.link(
5456 self.parser_functions.parse_inlines_no_link(label),
5457 r.url, r.title)
5458 else
5459 return fallback
5460 end
5461 end)
5462 end
5463
5464 -- lookup image reference and return an image, if the reference is found,
5465 -- or a bracketed label otherwise.
5466 local function indirect_image(label,sps,tag)
5467 return writer.defer_call(function()
5468 local r,fallback = lookup_reference(label,sps,tag)
5469 if r then
5470 return writer.image(writer.string(label), r.url, r.title)
5471 else
5472 return {"!", fallback}
5473 end
5474 end)
5475 end

```

### 3.1.5.7 Inline Elements (local)

```

5476 parsers.Str = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5477 / writer.string
5478
5479 parsers.Symbol = (V("SpecialChar") - parsers.tightblocksep)
5480 / writer.string
5481
5482 parsers.Ellipsis = P("...") / writer.ellipsis
5483
5484 parsers.Smart = parsers.Ellipsis

```



```

5485
5486 parsers.Code = parsers.inticks / writer.code
5487
5488 if options.blankBeforeBlockquote then
5489 parsers.bqstart = parsers.fail
5490 else
5491 parsers.bqstart = parsers.more
5492 end
5493
5494 if options.blankBeforeHeading then
5495 parsers.headerstart = parsers.fail
5496 else
5497 parsers.headerstart = parsers.hash
5498 + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5499 * parsers.optionalspace * parsers.newline)
5500 end
5501
5502 parsers.EndlineExceptions
5503 = parsers.blankline -- paragraph break
5504 + parsers.tightblocksep -- nested list
5505 + parsers.eof --end of document
5506 + parsers.bqstart
5507 + parsers.headerstart
5508
5509 parsers.Endline = parsers.newline
5510 * -V("EndlineExceptions")
5511 * parsers.spacechar^0
5512 / (options.hardLineBreaks and writer.linebreak
5513 or writer.space)
5514
5515 parsers.OptionalIndent
5516 = parsers.spacechar^1 / writer.space
5517
5518 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5519 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5520 + parsers.spacechar^1 * parsers.Endline
5521 * parsers.optionalspace
5522 / (options.hardLineBreaks
5523 and writer.linebreak
5524 or writer.space)
5525 + parsers.spacechar^1 * parsers.optionalspace
5526 / writer.space
5527
5528 parsers.NonbreakingEndline
5529 = parsers.newline
5530 * -V("EndlineExceptions")
5531 * parsers.spacechar^0

```

```

5532 / (options.hardLineBreaks and writer.linebreak
5533 or writer.nbsp)
5534
5535 parsers.NonbreakingSpace
5536 = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5537 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5538 + parsers.spacechar^1 * parsers.Endline
5539 * parsers.optionalspace
5540 / (options.hardLineBreaks
5541 and writer.linebreak
5542 or writer.nbsp)
5543 + parsers.spacechar^1 * parsers.optionalspace
5544 / writer.nbsp
5545
5546 if options.underscores then
5547 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
5548 parsers.doubleasterisks)
5549 + parsers.between(parsers.Inline, parsers.doubleunderscores,
5550 parsers.doubleunderscores)
5551) / writer.strong
5552
5553 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
5554 parsers.asterisk)
5555 + parsers.between(parsers.Inline, parsers.underscore,
5556 parsers.underscore)
5557) / writer.emphasis
5558 else
5559 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
5560 parsers.doubleasterisks)
5561) / writer.strong
5562
5563 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
5564 parsers.asterisk)
5565) / writer.emphasis
5566 end
5567
5568 parsers.AutoLinkUrl = parsers.less
5569 * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5570 * parsers.more
5571 / function(url)
5572 return writer.link(writer.escape(url), url)
5573 end
5574
5575 parsers.AutoLinkEmail = parsers.less
5576 * C((parsers.alphanumeric + S("-._+"))^1
5577 * P("@") * parsers.urlchar^1)
5578 * parsers.more

```

```

5579 / function(email)
5580 return writer.link(writer.escape(email),
5581 "mailto: "..email)
5582 end
5583
5584 parsers.AutoLinkRelativeReference
5585 = parsers.less
5586 * C(parsers.urlchar~1)
5587 * parsers.more
5588 / function(url)
5589 return writer.link(writer.escape(url), url)
5590 end
5591
5592 parsers.DirectLink = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5593 * parsers.spnl
5594 * parsers.lparent
5595 * (parsers.url + Cc("")) -- link can be empty [foo]()
5596 * parsers.optionaltitle
5597 * parsers.rparent
5598 / writer.link
5599
5600 parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
1
5601 / indirect_link
5602
5603 -- parse a link or image (direct or indirect)
5604 parsers.Link = parsers.DirectLink + parsers.IndirectLink
5605
5606 parsers.DirectImage = parsers.exclamation
5607 * (parsers.tag / self.parser_functions.parse_inlines)
5608 * parsers.spnl
5609 * parsers.lparent
5610 * (parsers.url + Cc("")) -- link can be empty [foo]()
5611 * parsers.optionaltitle
5612 * parsers.rparent
5613 / writer.image
5614
5615 parsers.IndirectImage = parsers.exclamation * parsers.tag
5616 * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5617
5618 parsers.Image = parsers.DirectImage + parsers.IndirectImage
5619
5620 -- avoid parsing long strings of * or _ as emph/strong
5621 parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
5622 / writer.string
5623
5624 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string

```

```

5625
5626 parsers.InlineHtml = parsers.emptyelt_any / writer.inline_html_tag
5627 + (parsers.htmlcomment / self.parser_functions.parse_inlines_no
5628 / writer.inline_html_comment
5629 + parsers.htmlinstruction
5630 + parsers.openelt_any / writer.inline_html_tag
5631 + parsers.closeelt_any / writer.inline_html_tag
5632
5633 parsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
5634 + parsers.decentity / entities.dec_entity / writer.string
5635 + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.8 Block Elements (local)

```

5636 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_nest
5637 / writer.block_html_comment
5638 + parsers.emptyelt_block / writer.block_html_element
5639 + parsers.openelt_exact("hr") / writer.block_html_element
5640 + parsers.in_matched_block_tags / writer.block_html_element
5641 + parsers.htmlinstruction
5642
5643 parsers.Verbatim = Cs((parsers.blanklines
5644 * ((parsers.indentedline - parsers.blankline))^1)^1
5645) / self.expandtabs / writer.verbatim
5646
5647 parsers.Blockquote = Cs(parsers.blockquote_body^1)
5648 / self.parser_functions.parse_blocks_nested
5649 / writer.blockquote
5650
5651 parsers.HorizontalRule = (parsers.lineof(parsers.asterisk)
5652 + parsers.lineof(parsers.dash)
5653 + parsers.lineof(parsers.underscore)
5654) / writer.hrule
5655
5656 parsers.Reference = parsers.define_reference_parser / register_link
5657
5658 parsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline^1)
5659 * (parsers.newline
5660 * (parsers.blankline^1
5661 + #parsers.hash
5662 + #(parsers.leader * parsers.more * parsers.space^-
5663 1)
5664 + parsers.eof
5665)
5666 + parsers.eof)
5667 / writer.paragraph

```

```

5668 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
5669 / writer.plain

```

### 3.1.5.9 Lists (local)

```

5670 parsers.starter = parsers.bullet + parsers.enumerator
5671
5672 if options.taskLists then
5673 parsers.tickbox = (parsers.ticked_box
5674 + parsers.halfticked_box
5675 + parsers.unticked_box
5676) / writer.tickbox
5677 else
5678 parsers.tickbox = parsers.fail
5679 end
5680
5681 -- we use \001 as a separator between a tight list item and a
5682 -- nested list under it.
5683 parsers.NestedList = Cs((parsers.optionallyindentedline
5684 - parsers.starter)^1)
5685 / function(a) return "\001"..a end
5686
5687 parsers.ListBlockLine = parsers.optionallyindentedline
5688 - parsers.blankline - (parsers.indent~-1
5689 * parsers.starter)
5690
5691 parsers.ListBlock = parsers.line * parsers.ListBlockLine^0
5692
5693 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5694 * parsers.ListBlock
5695
5696 parsers.TightListItem = function(starter)
5697 return -parsers.HorizontalRule
5698 * (Cs(starter / "" * parsers.tickbox~-1 * parsers.ListBlock * parsers.Nested
5699 1)
5700 / self.parser_functions.parse_blocks_nested)
5701 * -(parsers.blanklines * parsers.indent)
5702 end
5703
5704 parsers.LooseListItem = function(starter)
5705 return -parsers.HorizontalRule
5706 * Cs(starter / "" * parsers.tickbox~-1 * parsers.ListBlock * Cc("\n")
5707 * (parsers.NestedList + parsers.ListContinuationBlock^0)
5708 * (parsers.blanklines / "\n\n")
5709) / self.parser_functions.parse_blocks_nested
5710 end

```

```

5711 parsers.BulletList = (Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
5712 * parsers.skipblanklines * -parsers.bullet
5713 + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5714 * parsers.skipblanklines)
5715 / writer.bulletlist
5716
5717 local function ordered_list(items,tight,startnum)
5718 if options.startNumber then
5719 startnum = tonumber(startnum) or 1 -- fallback for '#'
5720 if startnum ~= nil then
5721 startnum = math.floor(startnum)
5722 end
5723 else
5724 startnum = nil
5725 end
5726 return writer.orderedlist(items,tight,startnum)
5727 end
5728
5729 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
5730 (Ct(parsers.TightListItem(Cb("listtype")))
5731 * parsers.TightListItem(parsers.enumerator)^0)
5732 * Cc(true) * parsers.skipblanklines * -parsers.enumerator
5733 + Ct(parsers.LooseListItem(Cb("listtype")))
5734 * parsers.LooseListItem(parsers.enumerator)^0)
5735 * Cc(false) * parsers.skipblanklines
5736) * Cb("listtype") / ordered_list

```

### 3.1.5.10 Blank (local)

```

5737 parsers.Blank = parsers.blankline / ""
5738 + parsers.Reference
5739 + (parsers.tightblocksep / "\n")

```

### 3.1.5.11 Headings (local)

```

5740 -- parse atx header
5741 parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
5742 * parsers.optionalspace
5743 * (C(parsers.line)
5744 / strip_atx_end
5745 / self.parser_functions.parse_inlines)
5746 * Cb("level")
5747 / writer.heading
5748
5749 parsers.SetextHeading = #(parsers.line * S("="))
5750 * Ct(parsers.linechar^1
5751 / self.parser_functions.parse_inlines)
5752 * parsers.newline

```

```

5753 * parsers.HeadingLevel
5754 * parsers.optionalspace
5755 * parsers.newline
5756 / writer.heading
5757
5758 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

**3.1.5.12 Syntax Specification** Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain TeX output.

```

5759 function self.finalize_grammar(extensions)

```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```

5760 local walkable_syntax = (function(global_walkable_syntax)
5761 local local_walkable_syntax = {}
5762 for lhs, rule in pairs(global_walkable_syntax) do
5763 local_walkable_syntax[lhs] = util.table_copy(rule)
5764 end
5765 return local_walkable_syntax
5766 end)(walkable_syntax)

```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax` [*left-hand side terminal symbol*] before, instead of, or after a right-hand-side terminal symbol.

```

5767 local current_extension_name = nil
5768 self.insert_pattern = function(selector, pattern, pattern_name)
5769 assert(pattern_name == nil or type(pattern_name) == "string")
5770 local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
5771 assert(lhs ~= nil,
5772 [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
5773 .. selector .. ["]]
5774 assert(walkable_syntax[lhs] ~= nil,
5775 [[Rule]] .. lhs .. [[-> ... does not exist in markdown grammar]])
5776 assert(pos == "before" or pos == "after" or pos == "instead of",
5777 [[Expected positional specifier "before", "after", or "instead of", not "]]
5778 .. pos .. ["]]
5779 local rule = walkable_syntax[lhs]
5780 local index = nil
5781 for current_index, current_rhs in ipairs(rule) do
5782 if type(current_rhs) == "string" and current_rhs == rhs then
5783 index = current_index

```

```

5784 if pos == "after" then
5785 index = index + 1
5786 end
5787 break
5788 end
5789 end
5790 assert(index ~= nil,
5791 [[Rule]] .. lhs .. [[->]] .. rhs
5792 .. [[does not exist in markdown grammar]])
5793 local accountable_pattern
5794 if current_extension_name then
5795 accountable_pattern = { pattern, current_extension_name, pattern_name }
5796 else
5797 assert(type(pattern) == "string",
5798 [[reader->insert_pattern() was called outside an extension with]]
5799 .. [[a PEG pattern instead of a rule name]])
5800 accountable_pattern = pattern
5801 end
5802 if pos == "instead of" then
5803 rule[index] = accountable_pattern
5804 else
5805 table.insert(rule, index, accountable_pattern)
5806 end
5807 end

```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```

5808 local syntax =
5809 { "Blocks",
5810
5811 Blocks = (V("ExpectedJekyllData")
5812 * (V("Blank")^0 / writer.interblocksep)
5813)^-1
5814 * V("Blank")^0
5815 * V("Block")^-1
5816 * (V("Blank")^0 / writer.interblocksep
5817 * V("Block"))^0
5818 * V("Blank")^0 * parsers.eof,
5819
5820 ExpectedJekyllData = parsers.fail,
5821
5822 Blank = parsers.Blank,
5823
5824 Blockquote = parsers.Blockquote,
5825 Verbatim = parsers.Verbatim,
5826 HorizontalRule = parsers.HorizontalRule,
5827 BulletList = parsers.BulletList,

```



```

5828 OrderedList = parsers.OrderedList,
5829 Heading = parsers.Heading,
5830 DisplayHtml = parsers.DisplayHtml,
5831 Paragraph = parsers.Paragraph,
5832 Plain = parsers.Plain,
5833 EndlineExceptions = parsers.EndlineExceptions,
5834
5835 Str = parsers.Str,
5836 Space = parsers.Space,
5837 OptionalIndent = parsers.OptionalIndent,
5838 Endline = parsers.Endline,
5839 UlOrStarLine = parsers.UlOrStarLine,
5840 Strong = parsers.Strong,
5841 Emph = parsers.Emph,
5842 Link = parsers.Link,
5843 Image = parsers.Image,
5844 Code = parsers.Code,
5845 AutoLinkUrl = parsers.AutoLinkUrl,
5846 AutoLinkEmail = parsers.AutoLinkEmail,
5847 AutoLinkRelativeReference
5848 = parsers.AutoLinkRelativeReference,
5849 InlineHtml = parsers.InlineHtml,
5850 HtmlEntity = parsers.HtmlEntity,
5851 EscapedChar = parsers.EscapedChar,
5852 Smart = parsers.Smart,
5853 Symbol = parsers.Symbol,
5854 SpecialChar = parsers.fail,
5855 }

```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a PEG pattern. The function (re)defines `walkable_syntax[left-hand side terminal symbol]` to be equal to pattern.

```

5856 self.update_rule = function(rule_name, pattern)
5857 assert(current_extension_name ~= nil)
5858 assert(syntax[rule_name] ~= nil,
5859 [[Rule]] .. rule_name .. [[-> ... does not exist in markdown grammar]])
5860 local accountable_pattern = { pattern, current_extension_name, rule_name }
5861 walkable_syntax[rule_name] = { accountable_pattern }
5862 end

```

Define a hash table of all characters with special meaning and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar of markdown.

```

5863 local special_characters = {}
5864 self.add_special_character = function(c)
5865 table.insert(special_characters, c)
5866 syntax.SpecialChar = S(table.concat(special_characters, ""))

```

```

5867 end
5868
5869 self.add_special_character("*")
5870 self.add_special_character("`")
5871 self.add_special_character("(")
5872 self.add_special_character("]")
5873 self.add_special_character("<")
5874 self.add_special_character("!")
5875 self.add_special_character("\\")

```

Apply syntax extensions.

```

5876 for _, extension in ipairs(extensions) do
5877 current_extension_name = extension.name
5878 extension.extend_writer(writer)
5879 extension.extend_reader(self)
5880 end
5881 current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

5882 if options.debugExtensions then
5883 local sorted_lhs = {}
5884 for lhs, _ in pairs(walkable_syntax) do
5885 table.insert(sorted_lhs, lhs)
5886 end
5887 table.sort(sorted_lhs)
5888
5889 local output_lines = {"{"}
5890 for lhs_index, lhs in ipairs(sorted_lhs) do
5891 local encoded_lhs = util.encode_json_string(lhs)
5892 table.insert(output_lines, [[]] .. encoded_lhs .. [[[: []]])
5893 local rule = walkable_syntax[lhs]
5894 for rhs_index, rhs in ipairs(rule) do
5895 local human_readable_rhs
5896 if type(rhs) == "string" then
5897 human_readable_rhs = rhs
5898 else
5899 local pattern_name
5900 if rhs[3] then
5901 pattern_name = rhs[3]
5902 else
5903 pattern_name = "Anonymous Pattern"
5904 end
5905 local extension_name = rhs[2]
5906 human_readable_rhs = pattern_name .. [[(]] .. extension_name .. [[)]]
5907 end
5908 local encoded_rhs = util.encode_json_string(human_readable_rhs)
5909 local output_line = [[]] .. encoded_rhs

```

```

5910 if rhs_index < #rule then
5911 output_line = output_line .. ", "
5912 end
5913 table.insert(output_lines, output_line)
5914 end
5915 local output_line = "]"
5916 if lhs_index < #sorted_lhs then
5917 output_line = output_line .. ", "
5918 end
5919 table.insert(output_lines, output_line)
5920 end
5921 table.insert(output_lines, "}")
5922
5923 local output = table.concat(output_lines, "\n")
5924 local output_filename = options.debugExtensionsFileName
5925 local output_file = assert(io.open(output_filename, "w"),
5926 [[Could not open file "]] .. output_filename .. [[" for writing]])
5927 assert(output_file:write(output))
5928 assert(output_file:close())
5929 end

```

Duplicate the `Inline` rule as `IndentedInline` with the right-hand-side terminal symbol `Space` replaced with `OptionalIndent`.

```

5930 walkable_syntax["IndentedInline"] = util.table_copy(
5931 walkable_syntax["Inline"])
5932 self.insert_pattern(
5933 "IndentedInline instead of Space",
5934 "OptionalIndent")

```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```

5935 for lhs, rule in pairs(walkable_syntax) do
5936 syntax[lhs] = parsers.fail
5937 for _, rhs in ipairs(rule) do
5938 local pattern

```

Although the interface of the `reader->insert_pattern` method does document this (see Section ??), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

5939 if type(rhs) == "string" then
5940 pattern = V(rhs)
5941 else
5942 pattern = rhs[1]
5943 if type(pattern) == "string" then
5944 pattern = V(pattern)
5945 end

```

```

5946 end
5947 syntax[lhs] = syntax[lhs] + pattern
5948 end
5949 end

```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```

5950 if options.underscores then
5951 self.add_special_character("_")
5952 end
5953
5954 if not options.codeSpans then
5955 syntax.Code = parsers.fail
5956 end
5957
5958 if not options.html then
5959 syntax.DisplayHtml = parsers.fail
5960 syntax.InlineHtml = parsers.fail
5961 syntax.HtmlEntity = parsers.fail
5962 else
5963 self.add_special_character("&")
5964 end
5965
5966 if options.preserveTabs then
5967 options.stripIndent = false
5968 end
5969
5970 if not options.smartEllipses then
5971 syntax.Smart = parsers.fail
5972 else
5973 self.add_special_character(".")
5974 end
5975
5976 if not options.relativeReferences then
5977 syntax.AutoLinkRelativeReference = parsers.fail
5978 end
5979
5980 local blocks_nested_t = util.table_copy(syntax)
5981 blocks_nested_t.ExpectedJekyllData = parsers.fail
5982 parsers.blocks_nested = Ct(blocks_nested_t)
5983
5984 parsers.blocks = Ct(syntax)
5985
5986 local inlines_t = util.table_copy(syntax)
5987 inlines_t[1] = "Inlines"
5988 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")

```

```

5989 parsers.inlines = Ct(inlines_t)
5990
5991 local inlines_no_link_t = util.table_copy(inlines_t)
5992 inlines_no_link_t.Link = parsers.fail
5993 parsers.inlines_no_link = Ct(inlines_no_link_t)
5994
5995 local inlines_no_inline_note_t = util.table_copy(inlines_t)
5996 inlines_no_inline_note_t.InlineNote = parsers.fail
5997 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
5998
5999 local inlines_no_html_t = util.table_copy(inlines_t)
6000 inlines_no_html_t.DisplayHtml = parsers.fail
6001 inlines_no_html_t.InlineHtml = parsers.fail
6002 inlines_no_html_t.HtmlEntity = parsers.fail
6003 parsers.inlines_no_html = Ct(inlines_no_html_t)
6004
6005 local inlines_nbsp_t = util.table_copy(inlines_t)
6006 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6007 inlines_nbsp_t.Space = parsers.NonbreakingSpace
6008 parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

Return a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

6009 return function(input)
6010 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 2.1.3). The `cacheDir` option is disregarded.

```

6011 local opt_string = {}
6012 for k, _ in pairs(defaultOptions) do
6013 local v = options[k]
6014 if type(v) == "table" then
6015 for _, i in ipairs(v) do
6016 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
6017 end
6018 elseif k ~= "cacheDir" then
6019 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6020 end
6021 end
6022 table.sort(opt_string)
6023 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
6024 local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain T<sub>E</sub>X output via the `writer->pack` method.

```

6025 local function convert(input)
6026 local document = self.parser_functions.parse_blocks(input)

```

```

6027 return util.ropetostring(writer.document(document))
6028 end
6029 if options.eagerCache or options.finalizeCache then
6030 local name = util.cache(options.cacheDir, input, salt, convert,
6031 ".md" .. writer.suffix)
6032 output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

6033 else
6034 output = convert(input)
6035 end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

6036 if options.finalizeCache then
6037 local file, mode
6038 if options.frozenCacheCounter > 0 then
6039 mode = "a"
6040 else
6041 mode = "w"
6042 end
6043 file = assert(io.open(options.frozenCacheFileName, mode),
6044 [[Could not open file]] .. options.frozenCacheFileName
6045 .. [[for writing]])
6046 assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname]]
6047 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
6048 .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
6049 assert(file:close())
6050 end
6051 return output
6052 end
6053 end
6054 return self
6055 end

```

### 3.1.6 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```

6056 M.extensions = {}

```

**3.1.6.1 Citations** The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the

syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```
6057 M.extensions.citations = function(citation_nbsps)
```

Define table `escaped_citation_chars` containing the characters to escape in citations.

```
6058 local escaped_citation_chars = {
6059 ["{"] = "\\markdownRendererLeftBrace{}",
6060 ["}"] = "\\markdownRendererRightBrace{}",
6061 [%] = "\\markdownRendererPercentSign{}",
6062 ["\\"] = "\\markdownRendererBackslash{}",
6063 [#] = "\\markdownRendererHash{}",
6064 }
6065 return {
6066 name = "built-in citations syntax extension",
6067 extend_writer = function(self)
6068 local options = self.options
6069
```

Use the `escaped_citation_chars` to create the `escape_citation` escaper functions.

```
6070 local escape_citation = util.escaper(
6071 escaped_citation_chars,
6072 self.escaped_minimal_strings)
```

Define `writer->citation` as a function that will transform an input citation name `c` to the output format. If `writer->hybrid` is `true`, use the `writer->escape_minimal` function. Otherwise, use the `escape_citation` function.

```
6073 if options.hybrid then
6074 self.citation = self.escape_minimal
6075 else
6076 self.citation = escape_citation
6077 end
```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, 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.

```

6078 function self.citations(text_cites, cites)
6079 local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
6080 "{", #cites, "}"}
6081 for _,cite in ipairs(cites) do
6082 buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6083 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
6084 end
6085 return buffer
6086 end
6087 end, extend_reader = function(self)
6088 local parsers = self.parsers
6089 local writer = self.writer
6090
6091 local citation_chars
6092 = parsers.alphanumeric
6093 + S("#$%&--+<>~/_")
6094
6095 local citation_name
6096 = Cs(parsers.dash~1) * parsers.at
6097 * Cs(citation_chars
6098 * (((citation_chars + parsers.internal_punctuation
6099 - parsers.comma - parsers.semicolon)
6100 * -#((parsers.internal_punctuation - parsers.comma
6101 - parsers.semicolon)^0
6102 * -(citation_chars + parsers.internal_punctuation
6103 - parsers.comma - parsers.semicolon)))^0
6104 * citation_chars)^-1)
6105
6106 local citation_body_prenote
6107 = Cs((parsers.alphanumeric^1
6108 + parsers.bracketed
6109 + parsers.inticks
6110 + (parsers.anyescaped
6111 - (parsers.rbracket + parsers.blankline^2))
6112 - (parsers.spnl * parsers.dash~1 * parsers.at))^0)
6113
6114 local citation_body_postnote
6115 = Cs((parsers.alphanumeric^1
6116 + parsers.bracketed
6117 + parsers.inticks
6118 + (parsers.anyescaped
6119 - (parsers.rbracket + parsers.semicolon
6120 + parsers.blankline^2))
6121 - (parsers.spnl * parsers.rbracket))^0)
6122
6123 local citation_body_chunk

```



```

6124 = citation_body_prenote
6125 * parsers.spnl * citation_name
6126 * (parsers.internal_punctuation - parsers.semicolon)^-
1
6127 * parsers.spnl * citation_body_postnote
6128
6129 local citation_body
6130 = citation_body_chunk
6131 * (parsers.semicolon * parsers.spnl
6132 * citation_body_chunk)^0
6133
6134 local citation_headless_body_postnote
6135 = Cs((parsers.alphanumeric^1
6136 + parsers.bracketed
6137 + parsers.inticks
6138 + (parsers.anyescaped
6139 - (parsers.rbracket + parsers.at
6140 + parsers.semicolon + parsers.blankline^2))
6141 - (parsers.spnl * parsers.rbracket))^0)
6142
6143 local citation_headless_body
6144 = citation_headless_body_postnote
6145 * (parsers.sp * parsers.semicolon * parsers.spnl
6146 * citation_body_chunk)^0
6147
6148 local citations
6149 = function(text_cites, raw_cites)
6150 local function normalize(str)
6151 if str == "" then
6152 str = nil
6153 else
6154 str = (citation_nbsps and
6155 self.parser_functions.parse_inlines_nbsp or
6156 self.parser_functions.parse_inlines)(str)
6157 end
6158 return str
6159 end
6160
6161 local cites = {}
6162 for i = 1,#raw_cites,4 do
6163 cites[#cites+1] = {
6164 prenote = normalize(raw_cites[i]),
6165 suppress_author = raw_cites[i+1] == "-",
6166 name = writer.citation(raw_cites[i+2]),
6167 postnote = normalize(raw_cites[i+3]),
6168 }
6169 end

```

```

6170 return writer.citations(text_cites, cites)
6171 end
6172
6173 local TextCitations
6174 = Ct((parsers.spnl
6175 * Cc("")
6176 * citation_name
6177 * ((parsers.spnl
6178 * parsers.lbracket
6179 * citation_headless_body
6180 * parsers.rbracket) + Cc("")))~1)
6181 / function(raw_cites)
6182 return citations(true, raw_cites)
6183 end
6184
6185 local ParenthesizedCitations
6186 = Ct((parsers.spnl
6187 * parsers.lbracket
6188 * citation_body
6189 * parsers.rbracket)^1)
6190 / function(raw_cites)
6191 return citations(false, raw_cites)
6192 end
6193
6194 local Citations = TextCitations + ParenthesizedCitations
6195
6196 self.insert_pattern("Inline after Emph",
6197 Citations, "Citations")
6198
6199 self.add_special_character("@")
6200 self.add_special_character("-")
6201 end
6202 }
6203 end

```

**3.1.6.2 Content Blocks** The `extensions.content_blocks` function implements the iA,Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```

6204 M.extensions.content_blocks = function(language_map)

```

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

```

6205 local languages_json = (function()

```

```

6206 local base, prev, curr
6207 for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
6208 local file = io.open(pathname, "r")
6209 if not file then goto continue end
6210 local input = assert(file:read("*a"))
6211 assert(file:close())
6212 local json = input:gsub('("[^\\n]-"):', '[%1]=')
6213 curr = load("_ENV = {}; return "..json)()
6214 if type(curr) == "table" then
6215 if base == nil then
6216 base = curr
6217 else
6218 setmetatable(prev, { __index = curr })
6219 end
6220 prev = curr
6221 end
6222 ::continue::
6223 end
6224 return base or {}
6225 end()
6226
6227 return {
6228 name = "built-in content_blocks syntax extension",
6229 extend_writer = function(self)

```

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.

```

6230 function self.contentblock(src,suf,type,tit)
6231 if not self.is_writing then return "" end
6232 src = src..".."..suf
6233 suf = suf:lower()
6234 if type == "onlineimage" then
6235 return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
6236 {"",self.string(src),"},"",
6237 {"",self.uri(src),"},"",
6238 {"",self.string(tit or ""),"},""}
6239 elseif languages_json[suf] then
6240 return {"\\markdownRendererContentBlockCode{"..suf.."},"",
6241 {"",self.string(languages_json[suf]),"},"",
6242 {"",self.string(src),"},"",
6243 {"",self.uri(src),"},"",
6244 {"",self.string(tit or ""),"},""}
6245 else
6246 return {"\\markdownRendererContentBlock{"..suf.."},"",
6247 {"",self.string(src),"},"",

```

```

6248 {"",self.uri(src),"}",
6249 {"",self.string(tit or ""),"}"}
6250 end
6251 end
6252 end, extend_reader = function(self)
6253 local parsers = self.parsers
6254 local writer = self.writer
6255
6256 local contentblock_tail
6257 = parsers.optionaltitle
6258 * (parsers.newline + parsers.eof)
6259
6260 -- case insensitive online image suffix:
6261 local onlineimagesuffix
6262 = (function(...)
6263 local parser = nil
6264 for _, suffix in ipairs({...}) do
6265 local pattern=nil
6266 for i=1,#suffix do
6267 local char=suffix:sub(i,i)
6268 char = S(char:lower()..char:upper())
6269 if pattern == nil then
6270 pattern = char
6271 else
6272 pattern = pattern * char
6273 end
6274 end
6275 if parser == nil then
6276 parser = pattern
6277 else
6278 parser = parser + pattern
6279 end
6280 end
6281 return parser
6282 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6283
6284 -- online image url for iA Writer content blocks with mandatory suffix,
6285 -- allowing nested brackets:
6286 local onlineimageurl
6287 = (parsers.less
6288 * Cs((parsers.anyescaped
6289 - parsers.more
6290 - #(parsers.period
6291 * onlineimagesuffix
6292 * parsers.more
6293 * contentblock_tail))^0)
6294 * parsers.period

```

```

6295 * Cs(onlineimagesuffix)
6296 * parsers.more
6297 + (Cs((parsers.inparens
6298 + (parsers.anyescaped
6299 - parsers.spacing
6300 - parsers.rparent
6301 - #(parsers.period
6302 * onlineimagesuffix
6303 * contentblock_tail))))^0)
6304 * parsers.period
6305 * Cs(onlineimagesuffix))
6306) * Cc("onlineimage")
6307
6308 -- filename for iA Writer content blocks with mandatory suffix:
6309 local localfilepath
6310 = parsers.slash
6311 * Cs((parsers.anyescaped
6312 - parsers.tab
6313 - parsers.newline
6314 - #(parsers.period
6315 * parsers.alphanumeric^1
6316 * contentblock_tail))^1)
6317 * parsers.period
6318 * Cs(parsers.alphanumeric^1)
6319 * Cc("localfile")
6320
6321 local ContentBlock
6322 = parsers.leader
6323 * (localfilepath + onlineimageurl)
6324 * contentblock_tail
6325 / writer.contentblock
6326
6327 self.insert_pattern("Block before Blockquote",
6328 ContentBlock, "ContentBlock")
6329 end
6330 }
6331 end

```

**3.1.6.3 Definition Lists** The `extensions.definition_lists` function implements the definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

6332 M.extensions.definition_lists = function(tight_lists)
6333 return {
6334 name = "built-in definition_lists syntax extension",
6335 extend_writer = function(self)

```

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.

```

6336 local function dlitem(term, defs)
6337 local retVal = {"\\markdownRenderDlItem{",term,""}
6338 for _, def in ipairs(defs) do
6339 retVal[#retVal+1] = {"\\markdownRenderDlDefinitionBegin ",def,
6340 "\\markdownRenderDlDefinitionEnd "}
6341 end
6342 retVal[#retVal+1] = "\\markdownRenderDlItemEnd "
6343 return retVal
6344 end
6345
6346 function self.definitionlist(items,tight)
6347 if not self.is_writing then return "" end
6348 local buffer = {}
6349 for _,item in ipairs(items) do
6350 buffer[#buffer + 1] = dlitem(item.term, item.definitions)
6351 end
6352 if tight and tight_lists then
6353 return {"\\markdownRenderDlBeginTight\\n", buffer,
6354 "\\n\\markdownRenderDlEndTight"}
6355 else
6356 return {"\\markdownRenderDlBegin\\n", buffer,
6357 "\\n\\markdownRenderDlEnd"}
6358 end
6359 end
6360 end, extend_reader = function(self)
6361 local parsers = self.parsers
6362 local writer = self.writer
6363
6364 local defstartchar = S("~:")
6365
6366 local defstart = (defstartchar * #parsers.spacing
6367 * (parsers.tab + parsers.space^-3)
6368 + parsers.space * defstartchar * #parsers.spacing
6369 * (parsers.tab + parsers.space^-2)
6370 + parsers.space * parsers.space * defstartchar
6371 * #parsers.spacing
6372 * (parsers.tab + parsers.space^-1)
6373 + parsers.space * parsers.space * parsers.space
6374 * defstartchar * #parsers.spacing
6375)
6376
6377 local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

```

```

6378
6379 local function definition_list_item(term, defs, _)
6380 return { term = self.parser_functions.parse_inlines(term),
6381 definitions = defs }
6382 end
6383
6384 local DefinitionListItemLoose
6385 = C(parsers.line) * parsers.skipblanklines
6386 * Ct((defstart
6387 * parsers.indented_blocks(dlchunk)
6388 / self.parser_functions.parse_blocks_nested)^1)
6389 * Cc(false) / definition_list_item
6390
6391 local DefinitionListItemTight
6392 = C(parsers.line)
6393 * Ct((defstart * dlchunk
6394 / self.parser_functions.parse_blocks_nested)^1)
6395 * Cc(true) / definition_list_item
6396
6397 local DefinitionList
6398 = (Ct(DefinitionListItemLoose^1) * Cc(false)
6399 + Ct(DefinitionListItemTight^1)
6400 * (parsers.skipblanklines
6401 * -DefinitionListItemLoose * Cc(true))
6402) / writer.definitionlist
6403
6404 self.insert_pattern("Block after Heading",
6405 DefinitionList, "DefinitionList")
6406 end
6407 }
6408 end

```

**3.1.6.4 Fenced Code** The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires between a paragraph and the following fenced code block.

```

6409 M.extensions.fenced_code = function(blank_before_code_fence)
6410 return {
6411 name = "built-in fenced_code syntax extension",
6412 extend_writer = function(self)
6413 local options = self.options
6414

```

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

```

6415 function self.fencedCode(i, s)

```

```

6416 if not self.is_writing then return "" end
6417 s = string.gsub(s, '[\r\n%s]*$', '')
6418 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6419 return {"\\markdownRendererInputFencedCode{"..name.."}{"..i.."}"}
6420 end
6421 end, extend_reader = function(self)
6422 local parsers = self.parsers
6423 local writer = self.writer
6424
6425 local function captures_geq_length(_,i,a,b)
6426 return #a >= #b and i
6427 end
6428
6429 local infostring = (parsers.linechar - (parsers.backtick
6430 + parsers.space^1 * (parsers.newline + parsers.eof)))^0
6431
6432 local fenceindent
6433 local fencehead = function(char)
6434 return C(parsers.nonindentspace) / function(s) fenceindent = #s end
6435 * Cg(char^3, "fencelength")
6436 * parsers.optionalspace * C(infostring)
6437 * parsers.optionalspace * (parsers.newline + parsers.eof)
6438 end
6439
6440 local fencetail = function(char)
6441 return parsers.nonindentspace
6442 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6443 * parsers.optionalspace * (parsers.newline + parsers.eof)
6444 + parsers.eof
6445 end
6446
6447 local fencedline = function(char)
6448 return C(parsers.line - fencetail(char))
6449 / function(s)
6450 local i = 1
6451 local remaining = fenceindent
6452 while true do
6453 local c = s:sub(i, i)
6454 if c == " " and remaining > 0 then
6455 remaining = remaining - 1
6456 i = i + 1
6457 elseif c == "\t" and remaining > 3 then
6458 remaining = remaining - 4
6459 i = i + 1
6460 else
6461 break
6462 end

```



```

6463 end
6464 return s:sub(i)
6465 end
6466 end
6467
6468 local TildeFencedCode
6469 = fencehead(parsers.tilde)
6470 * Cs(fencedline(parsers.tilde)^0)
6471 * fencetail(parsers.tilde)
6472
6473 local BacktickFencedCode
6474 = fencehead(parsers.backtick)
6475 * Cs(fencedline(parsers.backtick)^0)
6476 * fencetail(parsers.backtick)
6477
6478 local FencedCode = (TildeFencedCode
6479 + BacktickFencedCode)
6480 / function(infostring, code)
6481 return writer.fencedCode(writer.string(infostring),
6482 self.expandtabs(code))
6483 end
6484
6485 self.insert_pattern("Block after Verbatim",
6486 FencedCode, "FencedCode")
6487
6488 local fencestart
6489 if blank_before_code_fence then
6490 fencestart = parsers.fail
6491 else
6492 fencestart = fencehead(parsers.backtick)
6493 + fencehead(parsers.tilde)
6494 end
6495
6496 local EndlineExceptions = parsers.EndlineExceptions + fencestart
6497 self.update_rule("EndlineExceptions", EndlineExceptions)
6498
6499 self.add_special_character("~")
6500 end
6501 }
6502 end

```

**3.1.6.5 Footnotes** The `extensions.footnotes` function implements the Pandoc footnote and inline footnote syntax extensions. When the `footnote` parameter is `true`, the Pandoc footnote syntax extension will be enabled. When the `inline_footnotes` parameter is `true`, the Pandoc inline footnote syntax extension will be enabled.

```

6503 M.extensions.footnotes = function(footnotes, inline_footnotes)
6504 assert(footnotes or inline_footnotes)
6505 return {
6506 name = "built-in footnotes syntax extension",
6507 extend_writer = function(self)

```

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

```

6508 function self.note(s)
6509 return {"\\markdownRendererFootnote{",s,""}
6510 end
6511 end, extend_reader = function(self)
6512 local parsers = self.parsers
6513 local writer = self.writer
6514
6515 if inline_footnotes then
6516 local InlineNote
6517 = parsers.circumflex
6518 * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
6519 / writer.note
6520
6521 self.insert_pattern("Inline after Emph",
6522 InlineNote, "InlineNote")
6523 end
6524 if footnotes then
6525 local function strip_first_char(s)
6526 return s:sub(2)
6527 end
6528
6529 local RawNoteRef
6530 = #(parsers.lbracket * parsers.circumflex)
6531 * parsers.tag / strip_first_char
6532
6533 local rawnotes = {}
6534
6535 -- like indirect_link
6536 local function lookup_note(ref)
6537 return writer.defer_call(function()
6538 local found = rawnotes[self.normalize_tag(ref)]
6539 if found then
6540 return writer.note(
6541 self.parser_functions.parse_blocks_nested(found))
6542 else
6543 return {"[",
6544 self.parser_functions.parse_inlines("^" .. ref), "]" }
6545 end
6546 end)

```

```

6547 end
6548
6549 local function register_note(ref,rawnote)
6550 rawnotes[self.normalize_tag(ref)] = rawnote
6551 return ""
6552 end
6553
6554 local NoteRef = RawNoteRef / lookup_note
6555
6556 local NoteBlock
6557 = parsers.leader * RawNoteRef * parsers.colon
6558 * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6559 / register_note
6560
6561 local Blank = NoteBlock + parsers.Blank
6562 self.update_rule("Blank", Blank)
6563
6564 self.insert_pattern("Inline after Emph",
6565 NoteRef, "NoteRef")
6566 end
6567
6568 self.add_special_character("^")
6569 end
6570 }
6571 end

```

**3.1.6.6 Header Attributes** The `extensions.header_attributes` function implements a syntax extension that enables the assignment of HTML attributes to headings.

```

6572 M.extensions.header_attributes = function()
6573 return {
6574 name = "built-in header_attributes syntax extension",
6575 extend_writer = function()
6576 end, extend_reader = function(self)
6577 local parsers = self.parsers
6578 local writer = self.writer
6579
6580 parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
6581 * parsers.optionalspace
6582 * (C(((parsers.linechar
6583 - ((parsers.hash^1
6584 * parsers.optionalspace
6585 * parsers.HeadingAttributes^-1
6586 + parsers.HeadingAttributes)
6587 * parsers.optionalspace
6588 * parsers.newline))
6589 * (parsers.linechar

```

```

6590 - parsers.hash
6591 - parsers.lbrace)^0)^1)
6592 / self.parser_functions.parse_inlines)
6593 * Cg(Ct(parsers.newline
6594 + (parsers.hash^1
6595 * parsers.optionalspace
6596 * parsers.HeadingAttributes~-1
6597 + parsers.HeadingAttributes)
6598 * parsers.optionalspace
6599 * parsers.newline), "attributes")
6600 * Cb("level")
6601 * Cb("attributes")
6602 / writer.heading
6603
6604 parsers.SettextHeading = #(parsers.line * S("="))
6605 * (C(((parsers.linechar
6606 - (parsers.HeadingAttributes
6607 * parsers.optionalspace
6608 * parsers.newline))
6609 * (parsers.linechar
6610 - parsers.lbrace)^0)^1)
6611 / self.parser_functions.parse_inlines)
6612 * Cg(Ct(parsers.newline
6613 + (parsers.HeadingAttributes
6614 * parsers.optionalspace
6615 * parsers.newline)), "attributes")
6616 * parsers.HeadingLevel
6617 * Cb("attributes")
6618 * parsers.optionalspace
6619 * parsers.newline
6620 / writer.heading
6621
6622 local Heading = parsers.AtxHeading + parsers.SettextHeading
6623 self.update_rule("Heading", Heading)
6624 end
6625 }
6626 end

```

**3.1.6.7 YAML Metadata** The `extensions.jekyll_data` function implements the Pandoc `yml_metadata_block` syntax extension for entering metadata in YAML. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata

```

6627 M.extensions.jekyll_data = function(expect_jekyll_data)
6628 return {
6629 name = "built-in jekyll_data syntax extension",
6630 extend_writer = function(self)

```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```

6631 function self.jekyllData(d, t, p)
6632 if not self.is_writing then return "" end
6633
6634 local buf = {}
6635
6636 local keys = {}
6637 for k, _ in pairs(d) do
6638 table.insert(keys, k)
6639 end
6640 table.sort(keys)
6641
6642 if not p then
6643 table.insert(buf, "\\markdownRendererJekyllDataBegin")
6644 end
6645
6646 if #d > 0 then
6647 table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin")
6648 table.insert(buf, self.uri(p or "null"))
6649 table.insert(buf, "{")
6650 table.insert(buf, #keys)
6651 table.insert(buf, ")")
6652 else
6653 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin")
6654 table.insert(buf, self.uri(p or "null"))
6655 table.insert(buf, "{")
6656 table.insert(buf, #keys)
6657 table.insert(buf, ")")
6658 end
6659
6660 for _, k in ipairs(keys) do
6661 local v = d[k]
6662 local typ = type(v)
6663 k = tostring(k or "null")
6664 if typ == "table" and next(v) ~= nil then
6665 table.insert(
6666 buf,
6667 self.jekyllData(v, t, k)
6668)
6669 else
6670 k = self.uri(k)
6671 v = tostring(v)

```

```

6672 if typ == "boolean" then
6673 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6674 table.insert(buf, k)
6675 table.insert(buf, "{")
6676 table.insert(buf, v)
6677 table.insert(buf, "}")
6678 elseif typ == "number" then
6679 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6680 table.insert(buf, k)
6681 table.insert(buf, "{")
6682 table.insert(buf, v)
6683 table.insert(buf, "}")
6684 elseif typ == "string" then
6685 table.insert(buf, "\\markdownRendererJekyllDataString{")
6686 table.insert(buf, k)
6687 table.insert(buf, "{")
6688 table.insert(buf, t(v))
6689 table.insert(buf, "}")
6690 elseif typ == "table" then
6691 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
6692 table.insert(buf, k)
6693 table.insert(buf, "}")
6694 else
6695 error(format("Unexpected type %s for value of " ..
6696 "YAML key %s", typ, k))
6697 end
6698 end
6699 end
6700
6701 if #d > 0 then
6702 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6703 else
6704 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6705 end
6706
6707 if not p then
6708 table.insert(buf, "\\markdownRendererJekyllDataEnd")
6709 end
6710
6711 return buf
6712 end
6713 end, extend_reader = function(self)
6714 local parsers = self.parsers
6715 local writer = self.writer
6716
6717 local JekyllData
6718 = Cmt(C((parsers.line - P("---") - P("..."))^0)

```

```

6719 , function(s, i, text) -- luacheck: ignore s i
6720 local data
6721 local ran_ok, _ = pcall(function()
6722 local tinyyaml = require("markdown-tinyyaml")
6723 data = tinyyaml.parse(text, {timestamps=false})
6724 end)
6725 if ran_ok and data ~= nil then
6726 return true, writer.jekyllData(data, function(s)
6727 return self.parser_functions.parse_blocks_nested(s)
6728 end, nil)
6729 else
6730 return false
6731 end
6732 end
6733)
6734
6735 local UnexpectedJekyllData
6736 = P("----")
6737 * parsers.blankline / 0
6738 * #(-parsers.blankline) -- if followed by blank, it's an hrule
6739 * JekyllData
6740 * (P("----") + P("..."))
6741
6742 local ExpectedJekyllData
6743 = (P("----")
6744 * parsers.blankline / 0
6745 * #(-parsers.blankline) -- if followed by blank, it's an hrule
6746)^-1
6747 * JekyllData
6748 * (P("----") + P("..."))^-1
6749
6750 self.insert_pattern("Block before Blockquote",
6751 UnexpectedJekyllData, "UnexpectedJekyllData")
6752 if expect_jekyll_data then
6753 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
6754 end
6755 end
6756 }
6757 end

```

**3.1.6.8 Pipe Tables** The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (affectionately known as pipe tables). When the `table_captions` parameter is `true`, the function also implements the Pandoc `table_captions` syntax extension for table captions.

```

6758 M.extensions.pipe_tables = function(table_captions)
6759

```

```

6760 local function make_pipe_table_rectangular(rows)
6761 local num_columns = #rows[2]
6762 local rectangular_rows = {}
6763 for i = 1, #rows do
6764 local row = rows[i]
6765 local rectangular_row = {}
6766 for j = 1, num_columns do
6767 rectangular_row[j] = row[j] or ""
6768 end
6769 table.insert(rectangular_rows, rectangular_row)
6770 end
6771 return rectangular_rows
6772 end
6773
6774 local function pipe_table_row(allow_empty_first_column
6775 , nonempty_column
6776 , column_separator
6777 , column)
6778 local row_beginning
6779 if allow_empty_first_column then
6780 row_beginning = -- empty first column
6781 #(parsers.spacechar^4
6782 * column_separator)
6783 * parsers.optionalspace
6784 * column
6785 * parsers.optionalspace
6786 -- non-empty first column
6787 + parsers.nonindentspace
6788 * nonempty_column^-1
6789 * parsers.optionalspace
6790 else
6791 row_beginning = parsers.nonindentspace
6792 * nonempty_column^-1
6793 * parsers.optionalspace
6794 end
6795
6796 return Ct(row_beginning
6797 * (-- single column with no leading pipes
6798 #(column_separator
6799 * parsers.optionalspace
6800 * parsers.newline)
6801 * column_separator
6802 * parsers.optionalspace
6803 -- single column with leading pipes or
6804 -- more than a single column
6805 + (column_separator
6806 * parsers.optionalspace

```



```

6807 * column
6808 * parsers.optionalspace)^1
6809 * (column_separator
6810 * parsers.optionalspace)^-1))
6811 end
6812
6813 return {
6814 name = "built-in pipe_tables syntax extension",
6815 extend_writer = function(self)

```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```

6816 function self.table(rows, caption)
6817 if not self.is_writing then return "" end
6818 local buffer = {"\\markdownRendererTable{",
6819 caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}" }
6820 local temp = rows[2] -- put alignments on the first row
6821 rows[2] = rows[1]
6822 rows[1] = temp
6823 for i, row in ipairs(rows) do
6824 table.insert(buffer, "{")
6825 for _, column in ipairs(row) do
6826 if i > 1 then -- do not use braces for alignments
6827 table.insert(buffer, "{")
6828 end
6829 table.insert(buffer, column)
6830 if i > 1 then
6831 table.insert(buffer, "}")
6832 end
6833 end
6834 table.insert(buffer, "}")
6835 end
6836 return buffer
6837 end
6838 end, extend_reader = function(self)
6839 local parsers = self.parsers
6840 local writer = self.writer
6841
6842 local table_hline_separator = parsers.pipe + parsers.plus
6843
6844 local table_hline_column = (parsers.dash
6845 - #(parsers.dash
6846 * (parsers.spacechar
6847 + table_hline_separator
6848 + parsers.newline)))^1
6849 * (parsers.colon * Cc("r")

```

```

6850 + parsers.dash * Cc("d"))
6851 + parsers.colon
6852 * (parsers.dash
6853 - #(parsers.dash
6854 * (parsers.spacechar
6855 + table_hline_separator
6856 + parsers.newline)))^1
6857 * (parsers.colon * Cc("c")
6858 + parsers.dash * Cc("l"))
6859
6860 local table_hline = pipe_table_row(false
6861 , table_hline_column
6862 , table_hline_separator
6863 , table_hline_column)
6864
6865 local table_caption_beginning = parsers.skipblanklines
6866 * parsers.nonindentspace
6867 * (P("Table")^-1 * parsers.colon)
6868 * parsers.optionalspace
6869
6870 local table_row = pipe_table_row(true
6871 , (C((parsers.linechar - parsers.pipe)^1)
6872 / self.parser_functions.parse_inlines)
6873 , parsers.pipe
6874 , (C((parsers.linechar - parsers.pipe)^0)
6875 / self.parser_functions.parse_inlines))
6876
6877 local table_caption
6878 if table_captions then
6879 table_caption = #table_caption_beginning
6880 * table_caption_beginning
6881 * Ct(parsers.IndentedInline^1)
6882 * parsers.newline
6883 else
6884 table_caption = parsers.fail
6885 end
6886
6887 local PipeTable = Ct(table_row * parsers.newline
6888 * table_hline
6889 * (parsers.newline * table_row)^0)
6890 / make_pipe_table_rectangular
6891 * table_caption^-1
6892 / writer.table
6893
6894 self.insert_pattern("Block after Blockquote",
6895 PipeTable, "PipeTable")
6896 end

```

```

6897 }
6898 end

```

**3.1.6.9 Strike-Through** The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

6899 M.extensions.strike_through = function()
6900 return {
6901 name = "built-in strike_through syntax extension",
6902 extend_writer = function(self)

```

Define `writer->strike_through` as a function that will transform a strike-through span `s` of input text to the output format.

```

6903 function self.strike_through(s)
6904 return {"\\markdownRendererStrikeThrough{" ,s,""} }
6905 end
6906 end, extend_reader = function(self)
6907 local parsers = self.parsers
6908 local writer = self.writer
6909
6910 local StrikeThrough = (
6911 parsers.between(parsers.Inline, parsers.doubletildes,
6912 parsers.doubletildes)
6913) / writer.strike_through
6914
6915 self.insert_pattern("Inline after Emph",
6916 StrikeThrough, "StrikeThrough")
6917
6918 self.add_special_character("~")
6919 end
6920 }
6921 end

```

**3.1.6.10 Superscripts** The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

6922 M.extensions.superscripts = function()
6923 return {
6924 name = "built-in superscripts syntax extension",
6925 extend_writer = function(self)

```

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

```

6926 function self.superscript(s)
6927 return {"\\markdownRendererSuperscript{" ,s,""} }
6928 end
6929 end, extend_reader = function(self)
6930 local parsers = self.parsers

```

```

6931 local writer = self.writer
6932
6933 local Superscript = (
6934 parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
6935) / writer.superscript
6936
6937 self.insert_pattern("Inline after Emph",
6938 Superscript, "Superscript")
6939
6940 self.add_special_character("^")
6941 end
6942 }
6943 end

```

**3.1.6.11 Subscripts** The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

6944 M.extensions.subscripts = function()
6945 return {
6946 name = "built-in subscripts syntax extension",
6947 extend_writer = function(self)

```

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

```

6948 function self.subscript(s)
6949 return {"\\markdownRendererSubscript{"s,"}"}
6950 end
6951 end, extend_reader = function(self)
6952 local parsers = self.parsers
6953 local writer = self.writer
6954
6955 local Subscript = (
6956 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
6957) / writer.subscript
6958
6959 self.insert_pattern("Inline after Emph",
6960 Subscript, "Subscript")
6961
6962 self.add_special_character("~")
6963 end
6964 }
6965 end

```

**3.1.6.12 Fancy Lists** The `extensions.fancy_lists` function implements the Pandoc fancy list extension.

```

6966 M.extensions.fancy_lists = function()
6967 return {

```

```

6968 name = "built-in fancy_lists syntax extension",
6969 extend_writer = function(self)
6970 local options = self.options
6971

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,
  - `OneParen` – parentheses, and
  - `Period` – periods.

```

6972 function self.fancylist(items,tight,startnum,numstyle,numdelim)
6973 if not self.is_writing then return "" end
6974 local buffer = {}
6975 local num = startnum
6976 for _,item in ipairs(items) do
6977 buffer[#buffer + 1] = self.fancyitem(item,num)
6978 if num ~= nil then
6979 num = num + 1
6980 end
6981 end
6982 local contents = util.intersperse(buffer,"\n")
6983 if tight and options.tightLists then
6984 return {"\\markdownRendererFancy01BeginTight{",
6985 numstyle,"}{",numdelim,"}",contents,
6986 "\\n\\markdownRendererFancy01EndTight "}
6987 else
6988 return {"\\markdownRendererFancy01Begin{",
6989 numstyle,"}{",numdelim,"}",contents,
6990 "\\n\\markdownRendererFancy01End "}
6991 end
6992 end

```

Define `writer->fancyitem` as a function that will transform an input fancy ordered

list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

6993 function self.fancyitem(s,num)
6994 if num ~= nil then
6995 return {"\\markdownRendererFancy01ItemWithNumber{" ,num,"}",s,
6996 "\\markdownRendererFancy01ItemEnd "}
6997 else
6998 return {"\\markdownRendererFancy01Item ",s,"\\markdownRendererFancy01ItemEnd "
6999 end
7000 end
7001 end, extend_reader = function(self)
7002 local parsers = self.parsers
7003 local options = self.options
7004 local writer = self.writer
7005
7006 local label = parsers.dig + parsers.letter
7007 local numdelim = parsers.period + parsers.rparent
7008 local enumerator = C(label^3 * numdelim) * #parsers.spacing
7009 + C(label^2 * numdelim) * #parsers.spacing
7010 * (parsers.tab + parsers.space^1)
7011 + C(label * numdelim) * #parsers.spacing
7012 * (parsers.tab + parsers.space^2)
7013 + parsers.space * C(label^2 * numdelim)
7014 * #parsers.spacing
7015 + parsers.space * C(label * numdelim)
7016 * #parsers.spacing
7017 * (parsers.tab + parsers.space^1)
7018 + parsers.space * parsers.space * C(label^1
7019 * numdelim) * #parsers.spacing
7020 local starter = parsers.bullet + enumerator
7021
7022 local NestedList = Cs((parsers.optionallyindentedline
7023 - starter)^1)
7024 / function(a) return "\\001"..a end
7025
7026 local ListBlockLine = parsers.optionallyindentedline
7027 - parsers.blankline - (parsers.indent^-1
7028 * starter)
7029
7030 local ListBlock = parsers.line * ListBlockLine^0
7031
7032 local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
7033 * ListBlock
7034
7035 local TightListItem = function(starter)
7036 return -parsers.HorizontalRule

```

```

7037 * (Cs(starter / "" * parsers.tickbox~-1 * ListBlock * NestedList~-
1)
7038 / self.parser_functions.parse_blocks_nested)
7039 * -(parsers.blanklines * parsers.indent)
7040 end
7041
7042 local LooseListItem = function(starter)
7043 return -parsers.HorizontalRule
7044 * Cs(starter / "" * parsers.tickbox~-1 * ListBlock * Cc("\n")
7045 * (NestedList + ListContinuationBlock~0)
7046 * (parsers.blanklines / "\n\n")
7047) / self.parser_functions.parse_blocks_nested
7048 end
7049
7050 local function roman2number(roman)
7051 local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
7052 local numeral = 0
7053
7054 local i = 1
7055 local len = string.len(roman)
7056 while i < len do
7057 local z1, z2 = romans[string.sub(roman, i, i)], romans[string.sub(roman, i+1, i+1)]
7058 if z1 < z2 then
7059 numeral = numeral + (z2 - z1)
7060 i = i + 2
7061 else
7062 numeral = numeral + z1
7063 i = i + 1
7064 end
7065 end
7066 if i <= len then numeral = numeral + romans[string.sub(roman,i,i)] end
7067 return numeral
7068 end
7069
7070 local function sniffstyle(itemprefix)
7071 local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])?")
7072 local numdelim
7073 if delimend == "." then
7074 numdelim = "OneParen"
7075 elseif delimend == "." then
7076 numdelim = "Period"
7077 else
7078 numdelim = "Default"
7079 end
7080 numstr = numstr or itemprefix
7081
7082 local num

```

```

7083 num = numstr:match("^([IVXL]+)")
7084 if num then
7085 return roman2number(num), "UpperRoman", numdelim
7086 end
7087 num = numstr:match("^([ivxl]+)")
7088 if num then
7089 return roman2number(string.upper(num)), "LowerRoman", numdelim
7090 end
7091 num = numstr:match("^([A-Z])")
7092 if num then
7093 return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
7094 end
7095 num = numstr:match("^([a-z])")
7096 if num then
7097 return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7098 end
7099 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7100 end
7101
7102 local function fancylist(items,tight,start)
7103 local startnum, numstyle, numdelim = sniffstyle(start)
7104 return writer.fancylist(items,tight,
7105 options.startNumber and startnum,
7106 numstyle or "Decimal",
7107 numdelim or "Default")
7108 end
7109
7110 local FancyList = Cg(enumerator, "listtype") *
7111 (Ct(TightListItem(Cb("listtype")))
7112 * TightListItem(enumerator)^0)
7113 * Cc(true) * parsers.skipblanklines * -enumerator
7114 + Ct(LooseListItem(Cb("listtype")))
7115 * LooseListItem(enumerator)^0)
7116 * Cc(false) * parsers.skipblanklines
7117) * Cb("listtype") / fancylist
7118
7119 self:update_rule("OrderedList", FancyList)
7120 end
7121 }
7122 end

```

### 3.1.7 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section ??.

```

7123 function M.new(options)

```



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

```
7124 options = options or {}
7125 setmetatable(options, { __index = function (_, key)
7126 return defaultOptions[key] end })
```

Apply built-in syntax extensions based on `options`.

```
7127 local extensions = {}
7128
7129 if options.contentBlocks then
7130 local content_blocks_extension = M.extensions.content_blocks(
7131 options.contentBlocksLanguageMap)
7132 table.insert(extensions, content_blocks_extension)
7133 end
7134
7135 if options.definitionLists then
7136 local definition_lists_extension = M.extensions.definition_lists(
7137 options.tightLists)
7138 table.insert(extensions, definition_lists_extension)
7139 end
7140
7141 if options.fencedCode then
7142 local fenced_code_extension = M.extensions.fenced_code(
7143 options.blankBeforeCodeFence)
7144 table.insert(extensions, fenced_code_extension)
7145 end
7146
7147 if options.headerAttributes then
7148 local header_attributes_extension = M.extensions.header_attributes()
7149 table.insert(extensions, header_attributes_extension)
7150 end
7151
7152 if options.jekyllData then
7153 local jekyll_data_extension = M.extensions.jekyll_data(
7154 options.expectJekyllData)
7155 table.insert(extensions, jekyll_data_extension)
7156 end
7157
7158 if options.pipeTables then
7159 local pipe_tables_extension = M.extensions.pipe_tables(
7160 options.tableCaptions)
7161 table.insert(extensions, pipe_tables_extension)
7162 end
7163
7164 if options.strikeThrough then
7165 local strike_through_extension = M.extensions.strike_through()
7166 table.insert(extensions, strike_through_extension)
7167 end
```

```

7168
7169 if options.subscripts then
7170 local subscript_extension = M.extensions.subscripts()
7171 table.insert(extensions, subscript_extension)
7172 end
7173
7174 if options.superscripts then
7175 local superscript_extension = M.extensions.superscripts()
7176 table.insert(extensions, superscript_extension)
7177 end
7178
7179 if options.footnotes or options.inlineFootnotes then
7180 local footnotes_extension = M.extensions.footnotes(
7181 options.footnotes, options.inlineFootnotes)
7182 table.insert(extensions, footnotes_extension)
7183 end
7184
7185 if options.citations then
7186 local citations_extension = M.extensions.citations(options.citationNbsps)
7187 table.insert(extensions, citations_extension)
7188 end
7189
7190 if options.fancyLists then
7191 local fancy_lists_extension = M.extensions.fancy_lists()
7192 table.insert(extensions, fancy_lists_extension)
7193 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

7194 for _, user_extension_filename in ipairs(options.extensions) do
7195 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

7196 local pathname = util.lookup_files(filename)
7197 local input_file = assert(io.open(pathname, "r"),
7198 [[Could not open user-defined syntax extension "]]
7199 .. pathname .. [[" for reading]])
7200 local input = assert(input_file:read("*a"))
7201 assert(input_file:close())
7202 local user_extension, err = load([[
7203 local sandbox = {}
7204 setmetatable(sandbox, {__index = _G})
7205 _ENV = sandbox
7206]] .. input)()
7207 assert(user_extension,
7208 [[Failed to compile user-defined syntax extension "]]
7209 .. pathname .. [[:]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

7210 assert(user_extension.api_version ~= nil,
7211 [[User-defined syntax extension "]] .. pathname
7212 .. [" does not specify mandatory field "api_version"]])
7213 assert(type(user_extension.api_version) == "number",
7214 [[User-defined syntax extension "]] .. pathname
7215 .. [" specifies field "api_version" of type "]]
7216 .. type(user_extension.api_version)
7217 .. [" but "number" was expected]])
7218 assert(user_extension.api_version > 0
7219 and user_extension.api_version <= metadata.user_extension_api_version,
7220 [[User-defined syntax extension "]] .. pathname
7221 .. [" uses syntax extension API version "]]
7222 .. user_extension.api_version .. [[but markdown.lua]]
7223 .. metadata.version .. [[uses API version]]
7224 .. metadata.user_extension_api_version
7225 .. [[, which is incompatible]])
7226
7227 assert(user_extension.grammar_version ~= nil,
7228 [[User-defined syntax extension "]] .. pathname
7229 .. [" does not specify mandatory field "grammar_version"]])
7230 assert(type(user_extension.grammar_version) == "number",
7231 [[User-defined syntax extension "]] .. pathname
7232 .. [" specifies field "grammar_version" of type "]]
7233 .. type(user_extension.grammar_version)
7234 .. [" but "number" was expected]])
7235 assert(user_extension.grammar_version == metadata.grammar_version,
7236 [[User-defined syntax extension "]] .. pathname
7237 .. [" uses grammar version "]] .. user_extension.grammar_version
7238 .. [[but markdown.lua]] .. metadata.version
7239 .. [[uses grammar version]] .. metadata.grammar_version
7240 .. [[, which is incompatible]])
7241
7242 assert(user_extension.finalize_grammar ~= nil,
7243 [[User-defined syntax extension "]] .. pathname
7244 .. [" does not specify mandatory "finalize_grammar" field]])
7245 assert(type(user_extension.finalize_grammar) == "function",
7246 [[User-defined syntax extension "]] .. pathname
7247 .. [" specifies field "finalize_grammar" of type "]]
7248 .. type(user_extension.finalize_grammar)
7249 .. [" but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section ??.)

```

7250 local extension = {
7251 name = [[user-defined "]] .. pathname .. [[" syntax extension]],
7252 extend_reader = user_extension.finalize_grammar,
7253 extend_writer = function() end,

```

```

7254 }
7255 return extension
7256 end)(user_extension_filename)
7257 table.insert(extensions, user_extension)
7258 end

```

Produce and return a conversion function from markdown to plain TeX.

```

7259 local writer = M.writer.new(options)
7260 local reader = M.reader.new(writer, options)
7261 local convert = reader.finalize_grammar(extensions)
7262
7263 return convert
7264 end
7265
7266 return M

```

### 3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.6.

```

7267
7268 local input
7269 if input_filename then
7270 local input_file = assert(io.open(input_filename, "r"),
7271 [[Could not open file]] .. input_filename .. [[for reading]])
7272 input = assert(input_file:read("*a"))
7273 assert(input_file:close())
7274 else
7275 input = assert(io.read("*a"))
7276 end
7277

```

First, ensure that the `options.cacheDir` directory exists.

```

7278 local lfs = require("lfs")
7279 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7280 assert(lfs.mkdir(options["cacheDir"]))
7281 end
7282
7283 local ran_ok, kpse = pcall(require, "kpse")
7284 if ran_ok then kpse.set_program_name("luatex") end
7285 local md = require("markdown")

```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```

7286 if metadata.version ~= md.metadata.version then
7287 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7288 "markdown.lua " .. md.metadata.version .. ".")
7289 end

```

```
7290 local convert = md.new(options)
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
7291 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7292
7293 if output_filename then
7294 local output_file = assert(io.open(output_filename, "w"),
7295 [[Could not open file]] .. output_filename .. [[for writing]])
7296 assert(output_file:write(output))
7297 assert(output_file:close())
7298 else
7299 assert(io.write(output))
7300 end
```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X 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<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```
7301 \ifx\markdownInfo\undefined
7302 \def\markdownInfo#1{%
7303 \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
7304 \fi
7305 \ifx\markdownWarning\undefined
7306 \def\markdownWarning#1{%
7307 \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
7308 \fi
7309 \ifx\markdownError\undefined
7310 \def\markdownError#1#2{%
7311 \errhelp{#2.}%
7312 \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
7313 \fi
```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
7314 \def\markdownRendererInterblockSeparatorPrototype{\par}%
7315 \def\markdownRendererLineBreakPrototype{\hfil\break}%
7316 \let\markdownRendererEllipsisPrototype\dots
7317 \def\markdownRendererNbspPrototype{~}%
7318 \def\markdownRendererLeftBracePrototype{\char`\{}%
7319 \def\markdownRendererRightBracePrototype{\char`\}}%
```

```

7320 \def\markdownRendererDollarSignPrototype{\char`$}%
7321 \def\markdownRendererPercentSignPrototype{\char`\}%
7322 \def\markdownRendererAmpersandPrototype{\&}%
7323 \def\markdownRendererUnderscorePrototype{\char`_}%
7324 \def\markdownRendererHashPrototype{\char`\#}%
7325 \def\markdownRendererCircumflexPrototype{\char`^}%
7326 \def\markdownRendererBackslashPrototype{\char`\}%
7327 \def\markdownRendererTildePrototype{\char`~}%
7328 \def\markdownRendererPipePrototype{|}%
7329 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
7330 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
7331 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7332 \markdownInput{#3}}%
7333 \def\markdownRendererContentBlockOnlineImagePrototype{%
7334 \markdownRendererImage}%
7335 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
7336 \markdownRendererInputFencedCode{#3}{#2}}%
7337 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
7338 \def\markdownRendererUlBeginPrototype{}%
7339 \def\markdownRendererUlBeginTightPrototype{}%
7340 \def\markdownRendererUlItemPrototype{}%
7341 \def\markdownRendererUlItemEndPrototype{}%
7342 \def\markdownRendererUlEndPrototype{}%
7343 \def\markdownRendererUlEndTightPrototype{}%
7344 \def\markdownRendererOlBeginPrototype{}%
7345 \def\markdownRendererOlBeginTightPrototype{}%
7346 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
7347 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
7348 \def\markdownRendererOlItemPrototype{}%
7349 \def\markdownRendererOlItemWithNumberPrototype#1{}%
7350 \def\markdownRendererOlItemEndPrototype{}%
7351 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
7352 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
7353 \def\markdownRendererFancyOlItemEndPrototype{}%
7354 \def\markdownRendererOlEndPrototype{}%
7355 \def\markdownRendererOlEndTightPrototype{}%
7356 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
7357 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
7358 \def\markdownRendererDlBeginPrototype{}%
7359 \def\markdownRendererDlBeginTightPrototype{}%
7360 \def\markdownRendererDlItemPrototype#1{#1}%
7361 \def\markdownRendererDlItemEndPrototype{}%
7362 \def\markdownRendererDlDefinitionBeginPrototype{}%
7363 \def\markdownRendererDlDefinitionEndPrototype{\par}%
7364 \def\markdownRendererDlEndPrototype{}%
7365 \def\markdownRendererDlEndTightPrototype{}%
7366 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%

```

```

7367 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}}%
7368 \def\markdownRendererBlockQuoteBeginPrototype{\par\begin group\it}%
7369 \def\markdownRendererBlockQuoteEndPrototype{\end group\par}%
7370 \def\markdownRendererInputVerbatimPrototype#1{%
7371 \par{\tt\input#1\relax{}}\par}%
7372 \def\markdownRendererInputFencedCodePrototype#1#2{%
7373 \markdownRendererInputVerbatimPrototype{#1}}%
7374 \def\markdownRendererHeadingOnePrototype#1{#1}%
7375 \def\markdownRendererHeadingTwoPrototype#1{#1}%
7376 \def\markdownRendererHeadingThreePrototype#1{#1}%
7377 \def\markdownRendererHeadingFourPrototype#1{#1}%
7378 \def\markdownRendererHeadingFivePrototype#1{#1}%
7379 \def\markdownRendererHeadingSixPrototype#1{#1}%
7380 \def\markdownRendererHorizontalRulePrototype{}%
7381 \def\markdownRendererFootnotePrototype#1{#1}%
7382 \def\markdownRendererCitePrototype#1{}%
7383 \def\markdownRendererTextCitePrototype#1{}%
7384 \def\markdownRendererTickedBoxPrototype{[X]}%
7385 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
7386 \def\markdownRendererUntickedBoxPrototype{[]}%
7387 \def\markdownRendererStrikeThroughPrototype#1{#1}%
7388 \def\markdownRendererSuperscriptPrototype#1{#1}%
7389 \def\markdownRendererSubscriptPrototype#1{#1}%

```

**3.2.2.1 YAML Metadata Renderer Prototypes** To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_t1` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_t1` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_t1` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

7390 \ExplSyntaxOn
7391 \seq_new:N \g_@@_jekyll_data_datatypes_seq
7392 \tl_const:Nn \c_@@_jekyll_data_sequence_t1 { sequence }
7393 \tl_const:Nn \c_@@_jekyll_data_mapping_t1 { mapping }
7394 \tl_const:Nn \c_@@_jekyll_data_scalar_t1 { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```

7395 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
7396 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
7397 {
7398 \seq_if_empty:NF
7399 \g_@@_jekyll_data_datatypes_seq
7400 {
7401 \seq_get_right:NN
7402 \g_@@_jekyll_data_datatypes_seq
7403 \l_tmpa_tl

```

If we are currently in a sequence, we will put an asterisk (\*) instead of a key into `\g_@@_jekyll_data_wildcard_absolute_address_seq` to make it represent a *wildcard*. Keeping a wildcard instead of a precise address makes it easy for the users to react to *any* item of a sequence regardless of how many there are, which can often be useful.

```

7404 \str_if_eq:NNTF
7405 \l_tmpa_tl
7406 \c_@@_jekyll_data_sequence_tl
7407 {
7408 \seq_put_right:Nn
7409 \g_@@_jekyll_data_wildcard_absolute_address_seq
7410 { * }
7411 }
7412 {
7413 \seq_put_right:Nn
7414 \g_@@_jekyll_data_wildcard_absolute_address_seq
7415 { #1 }
7416 }
7417 }
7418 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.



For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```

7419 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
7420 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
7421 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
7422 {
7423 \seq_pop_left:NN #1 \l_tmpa_tl
7424 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
7425 \seq_put_left:NV #1 \l_tmpa_tl
7426 }
7427 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
7428 {
7429 \markdown_jekyll_data_concatenate_address:NN
7430 \g_@@_jekyll_data_wildcard_absolute_address_seq
7431 \g_@@_jekyll_data_wildcard_absolute_address_tl
7432 \seq_get_right:NN
7433 \g_@@_jekyll_data_wildcard_absolute_address_seq
7434 \g_@@_jekyll_data_wildcard_relative_address_tl
7435 }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```

7436 \cs_new:Nn \markdown_jekyll_data_push:nN
7437 {
7438 \markdown_jekyll_data_push_address_segment:n
7439 { #1 }
7440 \seq_put_right:NV
7441 \g_@@_jekyll_data_datatypes_seq
7442 #2
7443 \markdown_jekyll_data_update_address_tls:
7444 }
7445 \cs_new:Nn \markdown_jekyll_data_pop:
7446 {
7447 \seq_pop_right:NN
7448 \g_@@_jekyll_data_wildcard_absolute_address_seq
7449 \l_tmpa_tl
7450 \seq_pop_right:NN
7451 \g_@@_jekyll_data_datatypes_seq
7452 \l_tmpa_tl
```

```

7453 \markdown_jekyll_data_update_address_tls:
7454 }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

7455 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
7456 {
7457 \keys_set_known:nn
7458 { markdown/jekyllData }
7459 { { #1 } = { #2 } }
7460 }
7461 \cs_generate_variant:Nn
7462 \markdown_jekyll_data_set_keyval:nn
7463 { Vn }
7464 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
7465 {
7466 \markdown_jekyll_data_push:nN
7467 { #1 }
7468 \c_@@_jekyll_data_scalar_tl
7469 \markdown_jekyll_data_set_keyval:Vn
7470 \g_@@_jekyll_data_wildcard_absolute_address_tl
7471 { #2 }
7472 \markdown_jekyll_data_set_keyval:Vn
7473 \g_@@_jekyll_data_wildcard_relative_address_tl
7474 { #2 }
7475 \markdown_jekyll_data_pop:
7476 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

7477 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
7478 \markdown_jekyll_data_push:nN
7479 { #1 }
7480 \c_@@_jekyll_data_sequence_tl
7481 }
7482 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
7483 \markdown_jekyll_data_push:nN
7484 { #1 }
7485 \c_@@_jekyll_data_mapping_tl
7486 }
7487 \def\markdownRendererJekyllDataSequenceEndPrototype{
7488 \markdown_jekyll_data_pop:
7489 }
7490 \def\markdownRendererJekyllDataMappingEndPrototype{
7491 \markdown_jekyll_data_pop:
7492 }
7493 \def\markdownRendererJekyllDataBooleanPrototype#1#2{

```

```

7494 \markdown_jekyll_data_set_keyvals:nn
7495 { #1 }
7496 { #2 }
7497 }
7498 \def\markdownRendererJekyllDataEmptyPrototype#1{}
7499 \def\markdownRendererJekyllDataNumberPrototype#1#2{
7500 \markdown_jekyll_data_set_keyvals:nn
7501 { #1 }
7502 { #2 }
7503 }
7504 \def\markdownRendererJekyllDataStringPrototype#1#2{
7505 \markdown_jekyll_data_set_keyvals:nn
7506 { #1 }
7507 { #2 }
7508 }
7509 \ExplSyntaxOff

```

### 3.2.3 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain  $\text{\TeX}$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

7510 \ExplSyntaxOn
7511 \tl_new:N \g_@@_formatted_lua_options_tl
7512 \cs_new:Nn \@@_format_lua_options:
7513 {
7514 \tl_gclear:N
7515 \g_@@_formatted_lua_options_tl
7516 \seq_map_function:NN
7517 \g_@@_lua_options_seq
7518 \@@_format_lua_option:n
7519 }
7520 \cs_new:Nn \@@_format_lua_option:n
7521 {
7522 \@@_typecheck_option:n
7523 { #1 }
7524 \@@_get_option_type:nN
7525 { #1 }
7526 \l_tmpa_tl
7527 \bool_case_true:nF
7528 {
7529 {
7530 \str_if_eq_p:VV
7531 \l_tmpa_tl
7532 \c_@@_option_type_boolean_tl ||
7533 \str_if_eq_p:VV

```

```

7534 \l_tmpa_tl
7535 \c_@@_option_type_number_tl ||
7536 \str_if_eq_p:VV
7537 \l_tmpa_tl
7538 \c_@@_option_type_counter_tl
7539 }
7540 {
7541 \@@_get_option_value:nN
7542 { #1 }
7543 \l_tmpa_tl
7544 \tl_gput_right:Nx
7545 \g_@@_formatted_lua_options_tl
7546 { #1~== \l_tmpa_tl ,~ }
7547 }
7548 {
7549 \str_if_eq_p:VV
7550 \l_tmpa_tl
7551 \c_@@_option_type_clist_tl
7552 }
7553 {
7554 \@@_get_option_value:nN
7555 { #1 }
7556 \l_tmpa_tl
7557 \tl_gput_right:Nx
7558 \g_@@_formatted_lua_options_tl
7559 { #1~==\c_left_brace_str }
7560 \clist_map_inline:Vn
7561 \l_tmpa_tl
7562 {
7563 \tl_gput_right:Nx
7564 \g_@@_formatted_lua_options_tl
7565 { "##1" ,~ }
7566 }
7567 \tl_gput_right:Nx
7568 \g_@@_formatted_lua_options_tl
7569 { \c_right_brace_str ,~ }
7570 }
7571 }
7572 {
7573 \@@_get_option_value:nN
7574 { #1 }
7575 \l_tmpa_tl
7576 \tl_gput_right:Nx
7577 \g_@@_formatted_lua_options_tl
7578 { #1~== " \l_tmpa_tl " ,~ }
7579 }
7580 }

```

```

7581 \cs_generate_variant:Nn
7582 \clist_map_inline:nn
7583 { Vn }
7584 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
7585 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
7586 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```

7587 \def\markdownPrepare{%

```

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

```

7588 local lfs = require("lfs")
7589 local cacheDir = "\markdownOptionCacheDir"
7590 if not lfs.isdir(cacheDir) then
7591 assert(lfs.mkdir(cacheDir))
7592 end

```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

7593 local md = require("markdown")
7594 local convert = md.new(\markdownLuaOptions)
7595 }%

```

### 3.2.4 Buffering Markdown Input

The `\markdownIfOption{<name>}{<iftrue>}{<iffalse>}` macro is provided for testing, whether the value of `\markdownOption<name>` is `true`. If the value is `true`, then `<iftrue>` is expanded, otherwise `<iffalse>` is expanded.

```

7596 \ExplSyntaxOn
7597 \cs_new:Nn
7598 \@@_if_option:nTF
7599 {
7600 \@@_get_option_type:nN
7601 { #1 }
7602 \l_tmpa_tl
7603 \str_if_eq:NNF
7604 \l_tmpa_tl
7605 \c_@@_option_type_boolean_tl
7606 {
7607 \msg_error:nxxx
7608 { @@ }
7609 { expected-boolean-option }
7610 { #1 }
7611 { \l_tmpa_tl }
7612 }

```

```

7613 \@@_get_option_value:nN
7614 { #1 }
7615 \l_tmpa_tl
7616 \str_if_eq:NNTF
7617 \l_tmpa_tl
7618 \c_@@_option_value_true_tl
7619 { #2 }
7620 { #3 }
7621 }
7622 \msg_new:nnn
7623 { @@ }
7624 { expected-boolean-option }
7625 {
7626 Option~#1~has~type~#2,~
7627 but~a~boolean~was~expected.
7628 }
7629 \let\markdownIfOption=\@@_if_option:nTF
7630 \ExplSyntaxOff

```

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.

```

7631 \csname newread\endcsname\markdownInputFileStream
7632 \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

7633 \begingroup
7634 \catcode`\^^I=12%
7635 \gdef\markdownReadAndConvertTab{^^I}%
7636 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

7637 \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. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `\markdownOptionStripPercentSigns` is enabled.

```

7638 \catcode`\^^M=13%
7639 \catcode`\^^I=13%
7640 \catcode`=0%
7641 \catcode`\=12%
7642 |catcode`@=14%
7643 |catcode`|=12@
7644 |gdef|markdownReadAndConvert#1#2{
7645 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```

7646 |markdownIfOption{frozenCache}{@}{@
7647 |immediate|openout|markdownOutputFileStream@
7648 |markdownOptionInputTempFileName|relax@
7649 |markdownInfo{Buffering markdown input into the temporary @
7650 |input file "|markdownOptionInputTempFileName" and scanning @
7651 |for the closing token sequence "#1"}@
7652 }@

```

Locally change the category of the special plain T<sub>E</sub>X 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.

```

7653 |def|do##1{|catcode`##1=12}|dospecials@
7654 |catcode`| =12@
7655 |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `\markdownOptionStripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols ( $\sim$ ) are produced.

```

7656 |def|markdownReadAndConvertStripPercentSign##1{@
7657 |markdownIfOption{stripPercentSigns}{@
7658 |if##1%@
7659 |expandafter|expandafter|expandafter@
7660 |markdownReadAndConvertProcessLine@
7661 |else@
7662 |expandafter|expandafter|expandafter@
7663 |markdownReadAndConvertProcessLine@
7664 |expandafter|expandafter|expandafter##1@
7665 |fi@
7666 }{@
7667 |expandafter@
7668 |markdownReadAndConvertProcessLine@
7669 |expandafter##1@
7670 }@
7671 }@

```

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

```

7672 |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file. If we are reading markdown documents

from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

7673 |ifx|relax##3|relax@
7674 |markdownIfOption{frozenCache}{-}{@
7675 |immediate|write|markdownOutputFileStream{##1}@
7676 }@
7677 |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 T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

7678 |def^^M{@
7679 |markdownInfo{The ending token sequence was found}@
7680 |markdownIfOption{frozenCache}{-}{@
7681 |immediate|closeout|markdownOutputFileStream@
7682 }@
7683 |endgroup@
7684 |markdownInput{@
7685 |markdownOptionOutputDir@
7686 /|markdownOptionInputTempFileName@
7687 }@
7688 #2}@
7689 |fi@

```

Repeat with the next line.

```

7690 ^^M}@

```

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

```

7691 |catcode`|^I=13@
7692 |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.

```

7693 |catcode`|^M=13@
7694 |def^^M##1^^M{@
7695 |def^^M###1^^M{@
7696 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
7697 ^^M}@
7698 ^^M}@

```

Reset the character categories back to the former state.

```

7699 |endgroup

```



The following two sections of the implementation have been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```

7700 \ExplSyntaxOn
7701 \int_compare:nT
7702 { \markdownMode = 3 }
7703 {
7704 \markdownInfo{Using mode 3: ~The~lt3luabridge~package}
7705 \file_input:n { lt3luabridge.tex }
7706 \cs_new:Npn
7707 \markdownLuaExecute
7708 { \luabridgeExecute }
7709 }
7710 \ExplSyntaxOff

```

### 3.2.5 Lua Shell Escape Bridge

The following  $\TeX$  code is intended for  $\TeX$  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 Lua $\TeX$  engine, their  $\TeX$  distribution contains it, and uses shell access to produce and execute Lua scripts using the  $\TeX$  Lua interpreter [1, Section 4.1.1].

```

7711 \ifnum\markdownMode<2\relax
7712 \ifnum\markdownMode=0\relax
7713 \markdownWarning{Using mode 0: Shell escape via write18
7714 (deprecated, to be removed in Markdown 3.0.0)}%
7715 \else
7716 \markdownWarning{Using mode 1: Shell escape via os.execute
7717 (deprecated, to be removed in Markdown 3.0.0)}%
7718 \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` (Lua $\TeX$ , Pdf $\TeX$ ) or the `\shellescape` (X $\TeX$ ) 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.

```

7719 \ifx\pdfshellescape\undefined
7720 \ifx\shellescape\undefined
7721 \ifnum\markdownMode=0\relax
7722 \def\markdownExecuteShellEscape{1}%
7723 \else

```

```

7724 \def\markdownExecuteShellEscape{%
7725 \directlua{tex.sprint(status.shell_escape or "1")}}%
7726 \fi
7727 \else
7728 \let\markdownExecuteShellEscape\shellescape
7729 \fi
7730 \else
7731 \let\markdownExecuteShellEscape\pdfshellescape
7732 \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.

```

7733 \ifnum\markdownMode=0\relax
7734 \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
7735 \else
7736 \def\markdownExecuteDirect#1{%
7737 \directlua{os.execute("\luaescapestring{#1}")}}%
7738 \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.

```

7739 \def\markdownExecute#1{%
7740 \ifnum\markdownExecuteShellEscape=1\relax
7741 \markdownExecuteDirect{#1}%
7742 \else
7743 \markdownError{I can not access the shell}{Either run the TeX
7744 compiler with the --shell-escape or the --enable-write18 flag,
7745 or set shell_escape=t in the texmf.cnf file}%
7746 \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.

```

7747 \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.

```

7748 \catcode`\|=0%
7749 \catcode`\|=12%
7750 \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.

```

7751 \immediate\openout\markdownOutputFileStream=%
7752 \markdownOptionHelperScriptFileName

```

```

7753 |markdownInfo{Writing a helper Lua script to the file
7754 "|markdownOptionHelperScriptFileName"}%
7755 |immediate|write|markdownOutputFileStream{%
7756 local ran_ok, error = pcall(function()
7757 local ran_ok, kpse = pcall(require, "kpse")
7758 if ran_ok then kpse.set_program_name("luatex") end
7759 #1
7760 end)

```

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

```

7761 if not ran_ok then
7762 local file = io.open("%
7763 |markdownOptionOutputDir
7764 /|markdownOptionErrorTempFileName", "w")
7765 if file then
7766 file:write(error .. "\n")
7767 file:close()
7768 end
7769 print('\|markdownError{An error was encountered while executing
7770 Lua code}{For further clues, examine the file
7771 "|markdownOptionOutputDir
7772 /|markdownOptionErrorTempFileName"}')
7773 end}%
7774 |immediate|closeout|markdownOutputFileStream

```

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

```

7775 |markdownInfo{Executing a helper Lua script from the file
7776 "|markdownOptionHelperScriptFileName" and storing the result in the
7777 file "|markdownOptionOutputTempFileName"}%
7778 |markdownExecute{texlua "|markdownOptionOutputDir
7779 /|markdownOptionHelperScriptFileName" > %
7780 "|markdownOptionOutputDir
7781 /|markdownOptionOutputTempFileName"}%

```

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

```

7782 |input|markdownOptionOutputTempFileName|relax}%
7783 |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.

```

7784 \fi
7785 \ifnum\markdownMode=2\relax
7786 \markdownWarning{Using mode 2: Direct Lua access
7787 (deprecated, to be removed in Markdown 3.0.0)}%

```

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,

```

7788 \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.

```

7789 \catcode`\=0%
7790 \catcode`\=12%
7791 \gdef\markdownLuaExecute#1{%
7792 \directlua{%
7793 local function print(input)
7794 local output = {}
7795 for line in input:gmatch("[^\r\n]+") do
7796 table.insert(output, line)
7797 end
7798 tex.print(output)
7799 end
7800 #1
7801 }%
7802 }%
7803 \endgroup
7804 \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 T<sub>E</sub>X.

```

7805 \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. Furthermore, use the ampersand symbol to specify parameters.

```

7806 \catcode`\=0%
7807 \catcode`\=12%
7808 \catcode`\&=6%
7809 \gdef\markdownInput#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
7810 |begingroup
7811 |catcode`\%=12
```

Furthermore, also change the category code of the hash sign (#) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
7812 |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
7813 |markdownIfOption{frozenCache}{%
7814 |ifnum|markdownOptionFrozenCacheCounter=0|relax
7815 |markdownInfo{Reading frozen cache from
7816 |"|markdownOptionFrozenCacheFileName"}%
7817 |input|markdownOptionFrozenCacheFileName|relax
7818 |fi
7819 |markdownInfo{Including markdown document number
7820 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
7821 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
7822 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7823 }{%
7824 |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 L<sup>A</sup>T<sub>E</sub>X Mk to track changes to the markdown document.

```
7825 |openin|markdownInputFileStream&1
7826 |closein|markdownInputFileStream
7827 |markdownPrepareLuaOptions
7828 |markdownLuaExecute{%
7829 |markdownPrepare
7830 |local file = assert(io.open("&1", "r"),
7831 |[[Could not open file "&1" for reading]])
7832 |local input = assert(file:read("*a"))
7833 |assert(file:close())
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
7834 |print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%
```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```
7835 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7836 }%
7837 |endgroup
7838 }%
```

7839 |endgroup

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implemenation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [10, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```
7840 \def\markdownVersionSpace{ }%
7841 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
7842 \markdownVersion\markdownVersionSpace markdown renderer]%
```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```
7843 \ExplSyntaxOn
7844 \@@_latex_define_renderers:
7845 \@@_latex_define_renderer_prototypes:
7846 \ExplSyntaxOff
```

#### 3.3.1 Logging Facilities

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

#### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.2).

```
7847 \let\markdownInputPlainTeX\markdownInput
7848 \renewcommand\markdownInput[2][]{%
7849 \begingroup
7850 \markdownSetup{#1}%
7851 \markdownInputPlainTeX{#2}%
7852 \endgroup}%
```

The `markdown`, and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```
7853 \renewenvironment{markdown}{%
7854 \markdownReadAndConvert@markdown{}}{%
7855 \markdownEnd}%
7856 \renewenvironment{markdown*}[1]{%
7857 \markdownSetup{#1}%
7858 \markdownReadAndConvert@markdown*}{%
7859 \markdownEnd}%
7860 \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*.

```

7861 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
7862 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
7863 \gdef\markdownReadAndConvert@markdown#1<%
7864 \markdownReadAndConvert<\end{markdown#1}>%
7865 <|\end<markdown#1>>>%
7866 \endgroup

```

**3.3.2.1  $\text{\LaTeX}$  Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```

7867 \ExplSyntaxOn

```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```

7868 \newcommand\markdownLaTeXThemeName{}
7869 \seq_new:N \g_@@_latex_themes_seq
7870 \seq_put_right:NV
7871 \g_@@_latex_themes_seq
7872 \markdownLaTeXThemeName
7873 \newcommand\markdownLaTeXThemeLoad[2]{
7874 \def\@tempa{%
7875 \def\markdownLaTeXThemeName{#2}
7876 \seq_put_right:NV
7877 \g_@@_latex_themes_seq
7878 \markdownLaTeXThemeName
7879 \RequirePackage{#1}
7880 \seq_pop_right:NN
7881 \g_@@_latex_themes_seq
7882 \l_tmpa_tl
7883 \seq_get_right:NN
7884 \g_@@_latex_themes_seq
7885 \l_tmpa_tl
7886 \exp_args:NNV
7887 \def
7888 \markdownLaTeXThemeName
7889 \l_tmpa_tl}
7890 \ifmarkdownLaTeXLoaded
7891 \@tempa
7892 \else
7893 \exp_args:No
7894 \AtEndOfPackage
7895 { \@tempa }
7896 \fi}

```

7897 \ExplSyntaxOff

The `witiko/dot` theme enables the `fencedCode` Lua option:

7898 \markdownSetup{fencedCode}%

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

7899 \RequirePackage{ifthen,grffile}

We store the previous definition of the fenced code token renderer prototype:

7900 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype

7901 \markdownRendererInputFencedCodePrototype

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `\markdownOptionFrozenCache` plain  $\TeX$  option is disabled and the code block has not been previously typeset:

7902 \renewcommand\markdownRendererInputFencedCode[2]{%

7903 \def\next##1 ##2\relax{%

7904 \ifthenelse{\equal{##1}{dot}}{%

7905 \markdownIfOption{frozenCache}{}{%

7906 \immediate\write18{%

7907 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;

7908 then

7909 dot -Tpdf -o #1.pdf #1;

7910 cp #1 #1.pdf.source;

7911 fi}}%

We include the typeset image using the image token renderer:

7912 \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

7913 }{%

7914 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%

7915 }%

7916 }%

7917 \next#2 \relax}%

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

7918 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype

7919 \markdownRendererImagePrototype

We load the `catchfile` and `grffile` packages, see also Section 1.1.3:

7920 \RequirePackage{catchfile,grffile}

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.



```

7921 \newcount\markdown@witiko@graphicx@http@counter
7922 \markdown@witiko@graphicx@http@counter=0
7923 \newcommand\markdown@witiko@graphicx@http@filename{%
7924 \markdownOptionCacheDir/witiko_graphicx_http%
7925 .\the\markdown@witiko@graphicx@http@counter}%

```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```

7926 \newcommand\markdown@witiko@graphicx@http@download[2]{%
7927 wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}

```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```

7928 \begingroup
7929 \catcode`\%=12
7930 \catcode`\^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

7931 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
7932 \begingroup
7933 \edef\filename{\markdown@witiko@graphicx@http@filename}^^A

```

The image will be downloaded only if the image URL has the http or https protocols and the `\markdownOptionFrozenCache` plain TeX option is disabled:

```

7934 \markdownIfOption{frozenCache}{}{^^A
7935 \immediate\write18{^^A
7936 mkdir -p "\markdownOptionCacheDir";
7937 if printf '%s' "#3" | grep -q -E '^https?:';
7938 then

```

The image will be downloaded to the pathname `\markdownOptionCacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

7939 OUTPUT_PREFIX="\markdownOptionCacheDir";
7940 OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
7941 OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
7942 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

7943 if ! [-e "$OUTPUT"];
7944 then
7945 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
7946 printf '%s' "$OUTPUT" > "\filename";
7947 fi;

```

If the image does not have the http or https protocols or the image has already been downloaded, the URL will be stored as-is:

```

7948 else
7949 printf '%s' '#3' > "\filename";
7950 fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

7951 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
7952 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
7953 {#1}{#2}{\filename}{#4}^^A
7954 \endgroup
7955 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
7956 \endgroup

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

7957 \renewcommand\markdownRendererTildePrototype{~}%

```

### 3.3.3 Options

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

```

7958 \DeclareOption*{%
7959 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
7960 \ProcessOptions\relax

```

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

```

7961 \ExplSyntaxOn
7962 \keys_define:nn
7963 { markdown/latex-options }
7964 {
7965 renderers .code:n = {
7966 \keys_set:nn
7967 { markdown/latex-options/renderers }
7968 { #1 }
7969 },
7970 rendererPrototypes .code:n = {
7971 \keys_set:nn
7972 { markdown/latex-options/renderer-prototypes }
7973 { #1 }
7974 },

```

The `code` key is used to immediately expand and execute code, which can be especially useful in  $\text{\LaTeX}$  setup snippets.

```

7975 code .code:n = { #1 },

```

The `jeekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jeekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```

7976 jekyllDataRenderers .code:n = {
7977 \keys_set:nn
7978 { markdown/latex-options/jekyll-data-renderers }
7979 { #1 }
7980 },
7981 }
7982 \keys_define:nn
7983 { markdown/latex-options/jekyll-data-renderers }
7984 {
7985 unknown .code:n = {
7986 \tl_set_eq:NN
7987 \l_tmpa_tl
7988 \l_keys_key_str
7989 \tl_put_right:Nn
7990 \l_tmpa_tl
7991 {
7992 .code:n = { #1 }
7993 }
7994 \keys_define:nV
7995 { markdown/jekyllData }
7996 \l_tmpa_tl
7997 }
7998 }
7999 \cs_generate_variant:Nn
8000 \keys_define:nn
8001 { nV }
8002 \ExplSyntaxOff

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.1), none of it will take effect.

```

8003 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If the `tightLists` Lua option is disabled or the current document class is beamer, do not load the paralist package.

```

8004 \markdownIfOption{tightLists}{
8005 \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}}%
8006 }{}

```

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.

```

8007 \ExplSyntaxOn
8008 \@ifpackageloaded{paralist}{
8009 \tl_new:N
8010 \l_@@_latex_fancy_list_item_label_number_style_tl

```

```

8011 \tl_new:N
8012 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8013 \cs_new:Nn
8014 \@@_latex_fancy_list_item_label_number:nn
8015 {
8016 \str_case:nn
8017 { #1 }
8018 {
8019 { Decimal } { #2 }
8020 { LowerRoman } { \int_to_roman:n { #2 } }
8021 { UpperRoman } { \int_to_Roman:n { #2 } }
8022 { LowerAlpha } { \int_to_alph:n { #2 } }
8023 { UpperAlpha } { \int_to_alph:n { #2 } }
8024 }
8025 }
8026 \cs_new:Nn
8027 \@@_latex_fancy_list_item_label_delimiter:n
8028 {
8029 \str_case:nn
8030 { #1 }
8031 {
8032 { Default } { . }
8033 { OneParen } {) }
8034 { Period } { . }
8035 }
8036 }
8037 \cs_new:Nn
8038 \@@_latex_fancy_list_item_label:nnn
8039 {
8040 \@@_latex_fancy_list_item_label_number:nn
8041 { #1 }
8042 { #3 }
8043 \@@_latex_fancy_list_item_label_delimiter:n
8044 { #2 }
8045 }
8046 \cs_new:Nn
8047 \@@_latex_paralist_style:nn
8048 {
8049 \str_case:nn
8050 { #1 }
8051 {
8052 { Decimal } { 1 }
8053 { LowerRoman } { i }
8054 { UpperRoman } { I }
8055 { LowerAlpha } { a }
8056 { UpperAlpha } { A }
8057 }

```

```

8058 \@@_latex_fancy_list_item_label_delimiter:n
8059 { #2 }
8060 }
8061 \markdownSetup{rendererPrototypes={
8062 ulBeginTight = {\begin{compactitem}},
8063 ulEndTight = {\end{compactitem}},
8064 fancyOlBegin = {
8065 \group_begin:
8066 \tl_set:Nn
8067 \l_@@_latex_fancy_list_item_label_number_style_tl
8068 { #1 }
8069 \tl_set:Nn
8070 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8071 { #2 }
8072 \tl_set:Nn
8073 \l_tmpa_tl
8074 { \begin{enumerate}[] }
8075 \tl_put_right:Nx
8076 \l_tmpa_tl
8077 { \@@_latex_paralist_style:nn { #1 } { #2 } }
8078 \tl_put_right:Nn
8079 \l_tmpa_tl
8080 {] }
8081 \l_tmpa_tl
8082 },
8083 fancyOlEnd = {
8084 \end{enumerate}
8085 \group_end:
8086 },
8087 olBeginTight = {\begin{compactenum}},
8088 olEndTight = {\end{compactenum}},
8089 fancyOlBeginTight = {
8090 \group_begin:
8091 \tl_set:Nn
8092 \l_@@_latex_fancy_list_item_label_number_style_tl
8093 { #1 }
8094 \tl_set:Nn
8095 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8096 { #2 }
8097 \tl_set:Nn
8098 \l_tmpa_tl
8099 { \begin{compactenum}[] }
8100 \tl_put_right:Nx
8101 \l_tmpa_tl
8102 { \@@_latex_paralist_style:nn { #1 } { #2 } }
8103 \tl_put_right:Nn
8104 \l_tmpa_tl

```

```

8105 {] }
8106 \l_tmpa_tl
8107 },
8108 fancyOlEndTight = {
8109 \end{compactenum}
8110 \group_end:
8111 },
8112 fancyOlItemWithNumber = {
8113 \item
8114 [
8115 \@@_latex_fancy_list_item_label:VVn
8116 \l_@@_latex_fancy_list_item_label_number_style_tl
8117 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8118 { #1 }
8119]
8120 },
8121 dlBeginTight = {\begin{compactdesc}},
8122 dlEndTight = {\end{compactdesc}}}}
8123 \cs_generate_variant:Nn
8124 \@@_latex_fancy_list_item_label:nnn
8125 { VVn }
8126 }{
8127 \markdownSetup{rendererPrototypes={
8128 ulBeginTight = {\markdownRendererUlBegin},
8129 ulEndTight = {\markdownRendererUlEnd},
8130 fancyOlBegin = {\markdownRendererOlBegin},
8131 fancyOlEnd = {\markdownRendererOlEnd},
8132 olBeginTight = {\markdownRendererOlBegin},
8133 olEndTight = {\markdownRendererOlEnd},
8134 fancyOlBeginTight = {\markdownRendererOlBegin},
8135 fancyOlEndTight = {\markdownRendererOlEnd},
8136 dlBeginTight = {\markdownRendererDlBegin},
8137 dlEndTight = {\markdownRendererDlEnd}}}
8138 }
8139 \ExplSyntaxOff
8140 \RequirePackage{amsmath,ifthen}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

8141 \ifpackageloaded{unicode-math}{
8142 \markdownSetup{rendererPrototypes={
8143 untickedBox = {\mdlgwhtsquare$},
8144 }}
8145 }{
8146 \RequirePackage{amssymb}
8147 \markdownSetup{rendererPrototypes={
8148 untickedBox = {\square$},

```

```

8149 }}
8150 }
8151 \RequirePackage{csvsimple}
8152 \RequirePackage{fancyvrb}
8153 \RequirePackage{graphicx}
8154 \markdownSetup{rendererPrototypes={
8155 lineBreak = {\},
8156 leftBrace = {\textbraceleft},
8157 rightBrace = {\textbraceright},
8158 dollarSign = {\textdollar},
8159 underscore = {\textunderscore},
8160 circumflex = {\textasciicircum},
8161 backslash = {\textbackslash},
8162 tilde = {\textasciitilde},
8163 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>8</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

8164 codeSpan = {%
8165 \ifmmode
8166 \text{#1}%
8167 \else
8168 \texttt{#1}%
8169 \fi
8170 },
8171 contentBlock = {%
8172 \ifthenelse{\equal{#1}{csv}}{%
8173 \begin{table}%
8174 \begin{center}%
8175 \csvautotabular{#3}%
8176 \end{center}
8177 \ifx\empty#4\empty\else
8178 \caption{#4}%
8179 \fi
8180 \end{table}%
8181 }{%
8182 \ifthenelse{\equal{#1}{tex}}{%
8183 \catcode`\%=14\relax
8184 \catcode`\#=6\relax

```

---

<sup>8</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

8185 \input #3\relax
8186 \catcode`\%=12\relax
8187 \catcode`\#=12\relax
8188 }{%
8189 \markdownInput{#3}%
8190 }%
8191 }%
8192 },
8193 image = {%
8194 \begin{figure}%
8195 \begin{center}%
8196 \includegraphics{#3}%
8197 \end{center}%
8198 \ifx\empty#4\empty\else
8199 \caption{#4}%
8200 \fi
8201 \end{figure}},
8202 ulBegin = {\begin{itemize}},
8203 ulEnd = {\end{itemize}},
8204 olBegin = {\begin{enumerate}},
8205 olItem = {\item{}},
8206 olItemWithNumber = {\item[#1.]},
8207 olEnd = {\end{enumerate}},
8208 dlBegin = {\begin{description}},
8209 dlItem = {\item[#1]},
8210 dlEnd = {\end{description}},
8211 emphasis = {\emph{#1}},
8212 tickedBox = {\\boxtimes},
8213 halfTickedBox = {\\boxdot},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

8214 headerAttributeContextBegin = {
8215 \markdownSetup{
8216 rendererPrototypes = {
8217 attributeIdentifier = {%
8218 \begingroup
8219 \def\next####1{%
8220 \def####1#####1{%
8221 \endgroup
8222 #####1{#####1}%
8223 \label{##1}%
8224 }%
8225 }%
8226 \next\markdownRendererHeadingOne
8227 \next\markdownRendererHeadingTwo
8228 \next\markdownRendererHeadingThree

```



```

8229 \next\markdownRendererHeadingFour
8230 \next\markdownRendererHeadingFive
8231 \next\markdownRendererHeadingSix
8232 },
8233 },
8234 }%
8235 },
8236 superscript = {#1},
8237 subscript = {\textsubscript{#1}},
8238 blockQuoteBegin = {\begin{quotation}},
8239 blockQuoteEnd = {\end{quotation}},
8240 inputVerbatim = {\VerbatimInput{#1}},
8241 inputFencedCode = {%
8242 \ifx\relax#2\relax
8243 \VerbatimInput{#1}%
8244 \else
8245 \@ifundefined{minted@code}{%
8246 \@ifundefined{lst@version}{%
8247 \markdownRendererInputFencedCode{#1}{}%

```

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

```

8248 }{%
8249 \lstinputlisting[language=#2]{#1}%
8250 }%

```

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

```

8251 }{%
8252 \catcode`\#=6\relax
8253 \inputminted{#2}{#1}%
8254 \catcode`\#=12\relax
8255 }%
8256 \fi},
8257 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
8258 footnote = {\footnote{#1}}}}

```

Support the nesting of strong emphasis.

```

8259 \ExplSyntaxOn
8260 \def\markdownLATEXStrongEmphasis#1{%
8261 \str_if_in:NnTF
8262 \f@series
8263 { b }
8264 { \textnormal{#1} }
8265 { \textbf{#1} }
8266 }
8267 \ExplSyntaxOff
8268 \markdownSetup{rendererPrototypes={strongEmphasis={%
8269 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

8270 \@ifundefined{chapter}{%
8271 \markdownSetup{rendererPrototypes = {
8272 headingOne = {\section{#1}},
8273 headingTwo = {\subsection{#1}},
8274 headingThree = {\subsubsection{#1}},
8275 headingFour = {\paragraph{#1}\leavevmode},
8276 headingFive = {\subparagraph{#1}\leavevmode}}}
8277 }{%
8278 \markdownSetup{rendererPrototypes = {
8279 headingOne = {\chapter{#1}},
8280 headingTwo = {\section{#1}},
8281 headingThree = {\subsection{#1}},
8282 headingFour = {\subsubsection{#1}},
8283 headingFive = {\paragraph{#1}\leavevmode},
8284 headingSix = {\subparagraph{#1}\leavevmode}}}
8285 }%

```

**3.3.4.1 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

8286 \markdownSetup{
8287 rendererPrototypes = {
8288 ulItem = {%
8289 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUliItem
8290 },
8291 },
8292 }
8293 \def\markdownLaTeXUliItem{%
8294 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
8295 \item[\markdownLaTeXCheckbox]%
8296 \expandafter\@gobble
8297 \else
8298 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
8299 \item[\markdownLaTeXCheckbox]%
8300 \expandafter\expandafter\expandafter\@gobble
8301 \else
8302 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
8303 \item[\markdownLaTeXCheckbox]%
8304 \expandafter\expandafter\expandafter\expandafter
8305 \expandafter\expandafter\expandafter\@gobble
8306 \else
8307 \item{}%
8308 \fi
8309 \fi
8310 \fi
8311 }

```

**3.3.4.2 HTML elements** If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>9</sup>, we will pass HTML elements to the output HTML document unchanged.

```

8312 \@ifundefined{HCode}{-}{-}{
8313 \markdownSetup{
8314 rendererPrototypes = {
8315 inlineHtmlTag = {%
8316 \ifvmode
8317 \IgnorePar
8318 \EndP
8319 \fi
8320 \HCode{#1}%
8321 },
8322 inputBlockHtmlElement = {%
8323 \ifvmode
8324 \IgnorePar
8325 \fi
8326 \EndP
8327 \special{t4ht*<#1}%
8328 \par
8329 \ShowPar
8330 },
8331 },
8332 }
8333 }

```

**3.3.4.3 Citations** Here is a basic implementation for citations that uses the  $\text{\LaTeX}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

8334 \newcount\markdownLaTeXCitationsCounter
8335
8336 % Basic implementation
8337 \RequirePackage{gobble}
8338 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
8339 \advance\markdownLaTeXCitationsCounter by 1\relax
8340 \ifx\relax#4\relax
8341 \ifx\relax#5\relax
8342 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8343 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
8344 \expandafter\expandafter\expandafter
8345 \expandafter\expandafter\expandafter\expandafter
8346 \@gobblethree
8347 \fi
8348 \else% Before a postnote (#5), dump the accumulator

```

---

<sup>9</sup>See <https://tug.org/tex4ht/>.

```

8349 \ifx\relax#1\relax\else
8350 \cite{#1}%
8351 \fi
8352 \cite[#5]{#6}%
8353 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8354 \else
8355 \expandafter\expandafter\expandafter
8356 \expandafter\expandafter\expandafter\expandafter
8357 \expandafter\expandafter\expandafter
8358 \expandafter\expandafter\expandafter\expandafter
8359 \markdownLaTeXBasicCitations
8360 \fi
8361 \expandafter\expandafter\expandafter
8362 \expandafter\expandafter\expandafter\expandafter{%
8363 \expandafter\expandafter\expandafter
8364 \expandafter\expandafter\expandafter\expandafter}%
8365 \expandafter\expandafter\expandafter
8366 \expandafter\expandafter\expandafter\expandafter{%
8367 \expandafter\expandafter\expandafter
8368 \expandafter\expandafter\expandafter\expandafter}%
8369 \expandafter\expandafter\expandafter
8370 \@gobblethree
8371 \fi
8372 \else% Before a prenote (#4), dump the accumulator
8373 \ifx\relax#1\relax\else
8374 \cite{#1}%
8375 \fi
8376 \ifnum\markdownLaTeXCitationsCounter>1\relax
8377 \space % Insert a space before the prenote in later citations
8378 \fi
8379 #4-\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
8380 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8381 \else
8382 \expandafter\expandafter\expandafter
8383 \expandafter\expandafter\expandafter\expandafter
8384 \markdownLaTeXBasicCitations
8385 \fi
8386 \expandafter\expandafter\expandafter{%
8387 \expandafter\expandafter\expandafter}%
8388 \expandafter\expandafter\expandafter{%
8389 \expandafter\expandafter\expandafter}%
8390 \expandafter
8391 \@gobblethree
8392 \fi\markdownLaTeXBasicCitations{#1#2#6},}
8393 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
8394
8395 % Natbib implementation

```

```

8396 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
8397 \advance\markdownLaTeXCitationsCounter by 1\relax
8398 \ifx\relax#3\relax
8399 \ifx\relax#4\relax
8400 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8401 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
8402 \expandafter\expandafter\expandafter
8403 \expandafter\expandafter\expandafter\expandafter
8404 \@gobbletwo
8405 \fi
8406 \else% Before a postnote (#4), dump the accumulator
8407 \ifx\relax#1\relax\else
8408 \citep{#1}%
8409 \fi
8410 \citep[] [#4] {#5}%
8411 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8412 \else
8413 \expandafter\expandafter\expandafter
8414 \expandafter\expandafter\expandafter\expandafter
8415 \expandafter\expandafter\expandafter
8416 \expandafter\expandafter\expandafter\expandafter
8417 \markdownLaTeXNatbibCitations
8418 \fi
8419 \expandafter\expandafter\expandafter
8420 \expandafter\expandafter\expandafter\expandafter{%
8421 \expandafter\expandafter\expandafter
8422 \expandafter\expandafter\expandafter\expandafter}%
8423 \expandafter\expandafter\expandafter
8424 \@gobbletwo
8425 \fi
8426 \else% Before a prenote (#3), dump the accumulator
8427 \ifx\relax#1\relax\relax\else
8428 \citep{#1}%
8429 \fi
8430 \citep[#3] [#4] {#5}%
8431 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8432 \else
8433 \expandafter\expandafter\expandafter
8434 \expandafter\expandafter\expandafter\expandafter
8435 \markdownLaTeXNatbibCitations
8436 \fi
8437 \expandafter\expandafter\expandafter{%
8438 \expandafter\expandafter\expandafter}%
8439 \expandafter
8440 \@gobbletwo
8441 \fi\markdownLaTeXNatbibCitations{#1,#5}}
8442 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%

```

```

8443 \advance\markdownLaTeXCitationsCounter by 1\relax
8444 \ifx\relax#3\relax
8445 \ifx\relax#4\relax
8446 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8447 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
8448 \expandafter\expandafter\expandafter
8449 \expandafter\expandafter\expandafter\expandafter
8450 \@gobbletwo
8451 \fi
8452 \else% After a prenote or a postnote, dump the accumulator
8453 \ifx\relax#1\relax\else
8454 \citet{#1}%
8455 \fi
8456 , \citet[#3][#4]{#5}%
8457 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8458 ,
8459 \else
8460 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8461 ,
8462 \fi
8463 \fi
8464 \expandafter\expandafter\expandafter
8465 \expandafter\expandafter\expandafter\expandafter
8466 \markdownLaTeXNatbibTextCitations
8467 \expandafter\expandafter\expandafter
8468 \expandafter\expandafter\expandafter\expandafter{%
8469 \expandafter\expandafter\expandafter
8470 \expandafter\expandafter\expandafter\expandafter}%
8471 \expandafter\expandafter\expandafter
8472 \@gobbletwo
8473 \fi
8474 \else% After a prenote or a postnote, dump the accumulator
8475 \ifx\relax#1\relax\relax\else
8476 \citet{#1}%
8477 \fi
8478 , \citet[#3][#4]{#5}%
8479 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8480 ,
8481 \else
8482 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8483 ,
8484 \fi
8485 \fi
8486 \expandafter\expandafter\expandafter
8487 \markdownLaTeXNatbibTextCitations
8488 \expandafter\expandafter\expandafter{%
8489 \expandafter\expandafter\expandafter}%

```

```

8490 \expandafter
8491 \@gobbletwo
8492 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
8493
8494 % BibLaTeX implementation
8495 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
8496 \advance\markdownLaTeXCitationsCounter by 1\relax
8497 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8498 \autocites#1[#3][#4]{#5}%
8499 \expandafter\@gobbletwo
8500 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
8501 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
8502 \advance\markdownLaTeXCitationsCounter by 1\relax
8503 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8504 \textcites#1[#3][#4]{#5}%
8505 \expandafter\@gobbletwo
8506 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
8507
8508 \markdownSetup{rendererPrototypes = {
8509 cite = {%
8510 \markdownLaTeXCitationsCounter=1%
8511 \def\markdownLaTeXCitationsTotal{#1}%
8512 \@ifundefined{autocites}{%
8513 \ifundefined{citep}{%
8514 \expandafter\expandafter\expandafter
8515 \markdownLaTeXBasicCitations
8516 \expandafter\expandafter\expandafter{%
8517 \expandafter\expandafter\expandafter}%
8518 \expandafter\expandafter\expandafter{%
8519 \expandafter\expandafter\expandafter}%
8520 }{%
8521 \expandafter\expandafter\expandafter
8522 \markdownLaTeXNatbibCitations
8523 \expandafter\expandafter\expandafter{%
8524 \expandafter\expandafter\expandafter}%
8525 }%
8526 }{%
8527 \expandafter\expandafter\expandafter
8528 \markdownLaTeXBibLaTeXCitations
8529 \expandafter{\expandafter}%
8530 }},
8531 textCite = {%
8532 \markdownLaTeXCitationsCounter=1%
8533 \def\markdownLaTeXCitationsTotal{#1}%
8534 \@ifundefined{autocites}{%
8535 \ifundefined{citep}{%
8536 \expandafter\expandafter\expandafter

```

```

8537 \markdownLaTeXBasicTextCitations
8538 \expandafter\expandafter\expandafter{%
8539 \expandafter\expandafter\expandafter}%
8540 \expandafter\expandafter\expandafter{%
8541 \expandafter\expandafter\expandafter}%
8542 }{%
8543 \expandafter\expandafter\expandafter
8544 \markdownLaTeXNatbibTextCitations
8545 \expandafter\expandafter\expandafter{%
8546 \expandafter\expandafter\expandafter}%
8547 }%
8548 }{%
8549 \expandafter\expandafter\expandafter
8550 \markdownLaTeXBibLaTeXTextCitations
8551 \expandafter{\expandafter}%
8552 }}}

```

**3.3.4.4 Links** Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

8553 \RequirePackage{url}
8554 \RequirePackage{expl3}
8555 \ExplSyntaxOn
8556 \def\markdownRendererLinkPrototype#1#2#3#4{
8557 \tl_set:Nn \l_tmpa_tl { #1 }
8558 \tl_set:Nn \l_tmpb_tl { #2 }
8559 \bool_set:Nn
8560 \l_tmpa_bool
8561 {
8562 \tl_if_eq_p:NN
8563 \l_tmpa_tl
8564 \l_tmpb_tl
8565 }
8566 \tl_set:Nn \l_tmpa_tl { #4 }
8567 \bool_set:Nn
8568 \l_tmpb_bool
8569 {
8570 \tl_if_empty_p:N
8571 \l_tmpa_tl
8572 }

```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```

8573 \bool_if:nTF
8574 {

```



```

8575 \l_tmpa_bool && \l_tmpb_bool
8576 }
8577 {
8578 \markdownLaTeXRendererAutolink { #2 } { #3 }
8579 }{
8580 \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
8581 }
8582 }
8583 \def\markdownLaTeXRendererAutolink#1#2{%
 If the URL begins with a hash sign, then we assume that it is a relative reference.
 Otherwise, we assume that it is an absolute URL.

8584 \tl_set:Nn
8585 \l_tmpa_tl
8586 { #2 }
8587 \tl_trim_spaces:N
8588 \l_tmpa_tl
8589 \tl_set:Nx
8590 \l_tmpb_tl
8591 {
8592 \tl_range:Nnn
8593 \l_tmpa_tl
8594 { 1 }
8595 { 1 }
8596 }
8597 \str_if_eq:NNTF
8598 \l_tmpb_tl
8599 \c_hash_str
8600 {
8601 \tl_set:Nx
8602 \l_tmpb_tl
8603 {
8604 \tl_range:Nnn
8605 \l_tmpa_tl
8606 { 2 }
8607 { -1 }
8608 }
8609 \exp_args:NV
8610 \ref
8611 \l_tmpb_tl
8612 }{
8613 \url { #2 }
8614 }
8615 }
8616 \ExplSyntaxOff
8617 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
8618 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

**3.3.4.5 Tables** Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

8619 \newcount\markdownLaTeXRowCounter
8620 \newcount\markdownLaTeXRowTotal
8621 \newcount\markdownLaTeXColumnCounter
8622 \newcount\markdownLaTeXColumnTotal
8623 \newtoks\markdownLaTeXTable
8624 \newtoks\markdownLaTeXTableAlignment
8625 \newtoks\markdownLaTeXTableEnd
8626 \AtBeginDocument{%
8627 \ifpackageloaded{booktabs}{%
8628 \def\markdownLaTeXTopRule{\toprule}%
8629 \def\markdownLaTeXMidRule{\midrule}%
8630 \def\markdownLaTeXBottomRule{\bottomrule}%
8631 }{%
8632 \def\markdownLaTeXTopRule{\hline}%
8633 \def\markdownLaTeXMidRule{\hline}%
8634 \def\markdownLaTeXBottomRule{\hline}%
8635 }%
8636 }
8637 \markdownSetup{rendererPrototypes={
8638 table = {%
8639 \markdownLaTeXTable={}%
8640 \markdownLaTeXTableAlignment={}%
8641 \markdownLaTeXTableEnd={%
8642 \markdownLaTeXBottomRule
8643 \end{tabular}}}%
8644 \ifx\empty#1\empty\else
8645 \addto@hook\markdownLaTeXTable{%
8646 \begin{table}
8647 \centering}%
8648 \addto@hook\markdownLaTeXTableEnd{%
8649 \caption{#1}
8650 \end{table}}}%
8651 \fi
8652 \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
8653 \markdownLaTeXRowCounter=0%
8654 \markdownLaTeXRowTotal=#2%
8655 \markdownLaTeXColumnTotal=#3%
8656 \markdownLaTeXRenderTableRow
8657 }
8658 }}
8659 \def\markdownLaTeXRenderTableRow#1{%
8660 \markdownLaTeXColumnCounter=0%
8661 \ifnum\markdownLaTeXRowCounter=0\relax
8662 \markdownLaTeXReadAlignments#1%
8663 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%

```

```

8664 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
8665 \the\markdownLaTeXTableAlignment}}}%
8666 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
8667 \else
8668 \markdownLaTeXRenderTableCell#1%
8669 \fi
8670 \ifnum\markdownLaTeXRowCount=1\relax
8671 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
8672 \fi
8673 \advance\markdownLaTeXRowCount by 1\relax
8674 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
8675 \the\markdownLaTeXTable
8676 \the\markdownLaTeXTableEnd
8677 \expandafter\@gobble
8678 \fi\markdownLaTeXRenderTableRow}
8679 \def\markdownLaTeXReadAlignments#1{%
8680 \advance\markdownLaTeXColumnCounter by 1\relax
8681 \if#1d%
8682 \addto@hook\markdownLaTeXTableAlignment{1}%
8683 \else
8684 \addto@hook\markdownLaTeXTableAlignment{#1}%
8685 \fi
8686 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
8687 \expandafter\@gobble
8688 \fi\markdownLaTeXReadAlignments}
8689 \def\markdownLaTeXRenderTableCell#1{%
8690 \advance\markdownLaTeXColumnCounter by 1\relax
8691 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
8692 \addto@hook\markdownLaTeXTable{#1&}%
8693 \else
8694 \addto@hook\markdownLaTeXTable{#1\\}%
8695 \expandafter\@gobble
8696 \fi\markdownLaTeXRenderTableCell}
8697 \fi

```

**3.3.4.6 YAML Metadata** The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

8698 \ExplSyntaxOn
8699 \keys_define:nn
8700 { markdown/jekyllData }
8701 {
8702 author .code:n = { \author{#1} },
8703 date .code:n = { \date{#1} },
8704 title .code:n = { \title{#1} },
8705 }

```

To complement the default setup of our key-values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```

8706 % TODO: Remove the command definition in TeX Live 2021.
8707 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
8708 \markdownSetup{
8709 rendererPrototypes = {
8710 jekyllDataEnd = {
8711 % TODO: Remove the else branch in TeX Live 2021.
8712 \IfFormatAtLeastTF
8713 { 2020-10-01 }
8714 { \AddToHook{begindocument/end}{\maketitle} }
8715 {
8716 \ifx\@onlypreamble\@notprerr
8717 % We are in the document
8718 \maketitle
8719 \else
8720 % We are in the preamble
8721 \RequirePackage{etoolbox}
8722 \AfterEndPreamble{\maketitle}
8723 \fi
8724 }
8725 },
8726 },
8727 }
8728 \ExplSyntaxOff

```

**3.3.4.7 Strike-Through** If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

8729 \markdownIfOption{strikeThrough}{%
8730 \RequirePackage{soulutf8}%
8731 \markdownSetup{
8732 rendererPrototypes = {
8733 strikeThrough = {%
8734 \st{#1}%
8735 },
8736 }
8737 }
8738 }{}

```

### 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.

```
8739 \newcommand\markdownMakeOther{%
8740 \count0=128\relax
8741 \loop
8742 \catcode\count0=11\relax
8743 \advance\count0 by 1\relax
8744 \ifnum\count0<256\repeat}%
```

### 3.4 ConT<sub>E</sub>Xt Implementation

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

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` L<sup>A</sup>T<sub>E</sub>X package.

```
8745 \def\markdownMakeOther{%
8746 \count0=128\relax
8747 \loop
8748 \catcode\count0=11\relax
8749 \advance\count0 by 1\relax
8750 \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<sub>E</sub>Xt.

```
8751 \catcode`|=12}%
```

#### 3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```
8752 \long\def\inputmarkdown{%
8753 \dosingleempty
8754 \doinputmarkdown}%
8755 \long\def\doinputmarkdown[#1]#2{%
8756 \begingroup
8757 \iffirstargument
8758 \setupmarkdown{#1}%
8759 \fi
8760 \markdownInput{#2}%
8761 \endgroup}%
```

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

In Knuth’s  $\text{\TeX}$ , trailing spaces are removed very early on when a line is being put to the input buffer. [11, sec. 31]. According to Eijkhout [12, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

8762 \ifx\startluacode\undefined % MkII
8763 \begingroup
8764 \catcode`\|=0%
8765 \catcode`\|=12%
8766 |gdef|startmarkdown{%
8767 |markdownReadAndConvert{\stopmarkdown}%
8768 {|stopmarkdown}}%
8769 |gdef|stopmarkdown{%
8770 |markdownEnd}%
8771 |endgroup
8772 \else % MkIV
8773 \startluacode
8774 document.markdown_buffering = false
8775 local function preserve_trailing_spaces(line)
8776 if document.markdown_buffering then
8777 line = line:gsub("[\t][\t]$", "\t\t")
8778 end
8779 return line
8780 end
8781 resolvers.installinputlinehandler(preserve_trailing_spaces)
8782 \stopluacode
8783 \begingroup
8784 \catcode`\|=0%
8785 \catcode`\|=12%
8786 |gdef|startmarkdown{%
8787 |ctxlua{document.markdown_buffering = true}%
8788 |markdownReadAndConvert{\stopmarkdown}%
8789 {|stopmarkdown}}%
8790 |gdef|stopmarkdown{%
8791 |ctxlua{document.markdown_buffering = false}%
8792 |markdownEnd}%
8793 |endgroup
8794 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

8795 \def\markdownRendererLineBreakPrototype{\blank}%

```

```

8796 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
8797 \def\markdownRendererRightBracePrototype{\textbraceright}%
8798 \def\markdownRendererDollarSignPrototype{\textdollar}%
8799 \def\markdownRendererPercentSignPrototype{\percent}%
8800 \def\markdownRendererUnderscorePrototype{\textunderscore}%
8801 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
8802 \def\markdownRendererBackslashPrototype{\textbackslash}%
8803 \def\markdownRendererTildePrototype{\textasciitilde}%
8804 \def\markdownRendererPipePrototype{\char`|}%
8805 \def\markdownRendererLinkPrototype#1#2#3#4{%
8806 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
8807 \fi\tt<\hyphenatedurl{#3}>}}%
8808 \usemodule[database]
8809 \defineseparatedlist
8810 [MarkdownConTeXtCSV]
8811 [separator={,},
8812 before=\bTABLE,after=\eTABLE,
8813 first=\bTR,last=\eTR,
8814 left=\bTD,right=\eTD]
8815 \def\markdownConTeXtCSV{csv}
8816 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8817 \def\markdownConTeXtCSV@arg{#1}%
8818 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
8819 \placetable[] [tab:#1]{#4}{%
8820 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
8821 \else
8822 \markdownInput{#3}%
8823 \fi}%
8824 \def\markdownRendererImagePrototype#1#2#3#4{%
8825 \placefigure[] []{#4}{\externalfigure[#3]}}%
8826 \def\markdownRendererULBeginPrototype{\startitemize}%
8827 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
8828 \def\markdownRendererULItemPrototype{\item}%
8829 \def\markdownRendererULEndPrototype{\stopitemize}%
8830 \def\markdownRendererULEndTightPrototype{\stopitemize}%
8831 \def\markdownRendererOLBeginPrototype{\startitemize[n]}%
8832 \def\markdownRendererOLBeginTightPrototype{\startitemize[packed,n]}%
8833 \def\markdownRendererOLItemPrototype{\item}%
8834 \def\markdownRendererOLItemWithNumberPrototype#1{\sym{#1.}}%
8835 \def\markdownRendererOLEndPrototype{\stopitemize}%
8836 \def\markdownRendererOLEndTightPrototype{\stopitemize}%
8837 \definedescription
8838 [MarkdownConTeXtDlItemPrototype]
8839 [location=hanging,
8840 margin=standard,
8841 headstyle=bold]%
8842 \definestartstop

```

```

8843 [MarkdownConTeXtDlPrototype]
8844 [before=\blank,
8845 after=\blank]%
8846 \definestartstop
8847 [MarkdownConTeXtDlTightPrototype]
8848 [before=\blank\startpacked,
8849 after=\stoppacked\blank]%
8850 \def\markdownRendererDlBeginPrototype{%
8851 \startMarkdownConTeXtDlPrototype}%
8852 \def\markdownRendererDlBeginTightPrototype{%
8853 \startMarkdownConTeXtDlTightPrototype}%
8854 \def\markdownRendererDlItemPrototype#1{%
8855 \startMarkdownConTeXtDlItemPrototype{#1}}%
8856 \def\markdownRendererDlItemEndPrototype{%
8857 \stopMarkdownConTeXtDlItemPrototype}%
8858 \def\markdownRendererDlEndPrototype{%
8859 \stopMarkdownConTeXtDlPrototype}%
8860 \def\markdownRendererDlEndTightPrototype{%
8861 \stopMarkdownConTeXtDlTightPrototype}%
8862 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
8863 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
8864 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
8865 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
8866 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
8867 \def\markdownRendererInputFencedCodePrototype#1#2{%
8868 \ifx\relax#2\relax
8869 \typefile{#1}%
8870 \else

```

The code fence infostring is used as a name from the ConT<sub>E</sub>Xt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

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

```



```

8871 \typefile[#2] []{#1}%
8872 \fi}%
8873 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
8874 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
8875 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
8876 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
8877 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
8878 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
8879 \def\markdownRendererHorizontalRulePrototype{%
8880 \blackrule[height=1pt, width=\hsize]}%
8881 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
8882 \def\markdownRendererTickedBoxPrototype{\boxtimes$}%
8883 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}%
8884 \def\markdownRendererUntickedBoxPrototype{\square$}%
8885 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
8886 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
8887 \def\markdownRendererSubscriptPrototype#1{\low{#1}}

```

### 3.4.2.1 Tables

There is a basic implementation of tables.

```

8888 \newcount\markdownConTeXtRowCounter
8889 \newcount\markdownConTeXtRowTotal
8890 \newcount\markdownConTeXtColumnCounter
8891 \newcount\markdownConTeXtColumnTotal
8892 \newtoks\markdownConTeXtTable
8893 \newtoks\markdownConTeXtTableFloat
8894 \def\markdownRendererTablePrototype#1#2#3{%
8895 \markdownConTeXtTable={}%
8896 \ifx\empty#1\empty
8897 \markdownConTeXtTableFloat={%
8898 \the\markdownConTeXtTable}%
8899 \else
8900 \markdownConTeXtTableFloat={%
8901 \placetable{#1}{\the\markdownConTeXtTable}}%
8902 \fi
8903 \begingroup
8904 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8905 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8906 \setupTABLE[r][1][topframe=on, bottomframe=on]
8907 \setupTABLE[r][#1][bottomframe=on]
8908 \markdownConTeXtRowCounter=0%
8909 \markdownConTeXtRowTotal=#2%
8910 \markdownConTeXtColumnTotal=#3%
8911 \markdownConTeXtRenderTableRow}
8912 \def\markdownConTeXtRenderTableRow#1{%
8913 \markdownConTeXtColumnCounter=0%
8914 \ifnum\markdownConTeXtRowCounter=0\relax

```

```

8915 \markdownConTeXtReadAlignments#1%
8916 \markdownConTeXtTable={\bTABLE}%
8917 \else
8918 \markdownConTeXtTable=\expandafter{%
8919 \the\markdownConTeXtTable\bTR}%
8920 \markdownConTeXtRenderTableCell#1%
8921 \markdownConTeXtTable=\expandafter{%
8922 \the\markdownConTeXtTable\eTR}%
8923 \fi
8924 \advance\markdownConTeXtRowCounter by 1\relax
8925 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
8926 \markdownConTeXtTable=\expandafter{%
8927 \the\markdownConTeXtTable\eTABLE}%
8928 \the\markdownConTeXtTableFloat
8929 \endgroup
8930 \expandafter\gobbleoneargument
8931 \fi\markdownConTeXtRenderTableRow}
8932 \def\markdownConTeXtReadAlignments#1{%
8933 \advance\markdownConTeXtColumnCounter by 1\relax
8934 \if#1d%
8935 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8936 \fi\if#1l%
8937 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8938 \fi\if#1c%
8939 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
8940 \fi\if#1r%
8941 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
8942 \fi
8943 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8944 \expandafter\gobbleoneargument
8945 \fi\markdownConTeXtReadAlignments}
8946 \def\markdownConTeXtRenderTableCell#1{%
8947 \advance\markdownConTeXtColumnCounter by 1\relax
8948 \markdownConTeXtTable=\expandafter{%
8949 \the\markdownConTeXtTable\bTD#1\eTD}%
8950 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8951 \expandafter\gobbleoneargument
8952 \fi\markdownConTeXtRenderTableCell}
8953 \stopmodule\protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).

- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [5] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [6] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [7] Vít Novotný. *L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).
- [8] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [9] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [10] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [11] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [12] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

## Index

|                                    |                     |
|------------------------------------|---------------------|
| <code>\author</code>               | 259                 |
| <code>\autocites</code>            | 251                 |
| <code>blankBeforeBlockquote</code> | 15                  |
| <code>blankBeforeCodeFence</code>  | 16                  |
| <code>blankBeforeHeading</code>    | 16                  |
| <code>breakableBlockquotes</code>  | 16                  |
| <code>cacheDir</code>              | 14, 20, 41, 42, 189 |

|                              |                |
|------------------------------|----------------|
| citationNbsps                | 17             |
| citations                    | 17, 72         |
| \cite                        | 251            |
| \citep                       | 251            |
| \citet                       | 251            |
| codeSpans                    | 18             |
| compactdesc                  | 4              |
| compactenum                  | 4              |
| compactitem                  | 4              |
| contentBlocks                | 14, 18         |
| contentBlocksLanguageMap     | 14             |
| convert                      | 229            |
| \csvautotabular              | 4              |
| \date                        | 259            |
| debugExtensions              | 8, 15, 19, 186 |
| debugExtensionsFileName      | 15, 19         |
| defaultOptions               | 9, 37, 217     |
| \definetyping                | 264            |
| definitionLists              | 19, 61         |
| \directlua                   | 4, 42, 236     |
| eagerCache                   | 20, 20         |
| \enableregime                | 261            |
| \endmarkdown                 | 82             |
| entities.char_entity         | 149            |
| entities.dec_entity          | 149            |
| entities.hex_entity          | 149            |
| escape_citation              | 191, 191       |
| escaped_citation_chars       | 191, 191       |
| expandtabs                   | 172            |
| expectJekyllData             | 22             |
| extensions                   | 20, 100, 190   |
| extensions.citations         | 190            |
| extensions.content_blocks    | 194            |
| extensions.definition_lists  | 197            |
| extensions.fancy_lists       | 212            |
| extensions.fenced_code       | 199            |
| extensions.footnotes         | 201            |
| extensions.header_attributes | 203            |
| extensions.jekyll_data       | 204            |
| extensions.pipe_table        | 207            |

|                                                           |                                       |
|-----------------------------------------------------------|---------------------------------------|
| <code>extensions.strike_through</code>                    | 211                                   |
| <code>extensions.subscripts</code>                        | 212                                   |
| <code>extensions.superscripts</code>                      | 211                                   |
| <code>fancyLists</code>                                   | 23, 56–61                             |
| <code>fencedCode</code>                                   | 23, 65, 240                           |
| <code>\filecontents</code>                                | 230                                   |
| <code>finalizeCache</code>                                | 15, 20, 24, 25, 41, 190               |
| <code>footnotes</code>                                    | 24, 71                                |
| <code>frozenCacheCounter</code>                           | 25, 190, 237                          |
| <code>frozenCacheFileName</code>                          | 15, 24, 190                           |
| <code>\g_luabridge_error_output_filename_str</code>       | 43                                    |
| <code>\g_luabridge_helper_script_filename_str</code>      | 42                                    |
| <code>hardLineBreaks</code>                               | 25                                    |
| <code>hashEnumerators</code>                              | 26                                    |
| <code>headerAttributes</code>                             | 26, 30, 75, 76                        |
| <code>html</code>                                         | 27, 73, 74, 251                       |
| <code>hybrid</code>                                       | 27, 34, 35, 44, 49, 88, 152, 173, 237 |
| <code>\includegraphics</code>                             | 4                                     |
| <code>inlineFootnotes</code>                              | 27                                    |
| <code>\input</code>                                       | 39, 97, 232, 235                      |
| <code>\inputmarkdown</code>                               | 97, 98, 261                           |
| <code>isdir</code>                                        | 3                                     |
| <code>iterlines</code>                                    | 172                                   |
| <code>jekyllData</code>                                   | 3, 22, 28, 66–69                      |
| <code>\jobname</code>                                     | 42, 43                                |
| <code>\label</code>                                       | 248                                   |
| <code>languages_json</code>                               | 194, 194                              |
| <code>\maketitle</code>                                   | 260                                   |
| <code>\markdown</code>                                    | 82                                    |
| <code>markdown</code>                                     | 81, 81, 82, 238                       |
| <code>markdown*</code>                                    | 81, 81–83, 238                        |
| <code>\markdown_jekyll_data_concatenate_address:NN</code> | 225                                   |
| <code>\markdown_jekyll_data_pop:</code>                   | 225                                   |
| <code>\markdown_jekyll_data_push:nN</code>                | 225                                   |
| <code>\markdown_jekyll_data_push_address_segment:n</code> | 223                                   |
| <code>\markdown_jekyll_data_set_keyval:Nn</code>          | 226                                   |
| <code>\markdown_jekyll_data_set_keyvals:nn</code>         | 226                                   |

|                                                        |                                      |
|--------------------------------------------------------|--------------------------------------|
| <code>\markdown_jekyll_data_update_address_tls:</code> | 225                                  |
| <code>\markdownBegin</code>                            | 39, 39, 40, 79, 82, 97               |
| <code>\markdownEnd</code>                              | 39, 39, 40, 79, 82, 97               |
| <code>\markdownError</code>                            | 79, 79                               |
| <code>\markdownExecute</code>                          | 234                                  |
| <code>\markdownExecuteDirect</code>                    | 234, 234                             |
| <code>\markdownExecuteShellEscape</code>               | 233, 234                             |
| <code>\markdownIfOption</code>                         | 229                                  |
| <code>\markdownIfSnippetExists</code>                  | 83                                   |
| <code>\markdownInfo</code>                             | 79                                   |
| <code>\markdownInput</code>                            | 39, 40, 81–83, 98, 236, 238          |
| <code>\markdownInputFileStream</code>                  | 230                                  |
| <code>\markdownInputPlainTeX</code>                    | 238                                  |
| <code>\markdownLuaExecute</code>                       | 233, 234, 235, 236                   |
| <code>\markdownLuaOptions</code>                       | 227, 229                             |
| <code>\markdownLuaRegisterIBCallback</code>            | 81                                   |
| <code>\markdownLuaUnregisterIBCallback</code>          | 81                                   |
| <code>\markdownMakeOther</code>                        | 79, 261                              |
| <code>\markdownMode</code>                             | 4, 42, 43, 80, 80, 233, 235          |
| <code>\markdownOptionCacheDir</code>                   | 3, 94, 229, 241                      |
| <code>\markdownOptionErrorTempFileName</code>          | 43, 43, 235                          |
| <code>\markdownOptionFinalizeCache</code>              | 41, 41, 42, 93                       |
| <code>\markdownOptionFrozenCache</code>                | 15, 24, 41, 42, 87, 88, 93, 240, 241 |
| <code>\markdownOptionFrozenCacheFileName</code>        | 41                                   |
| <code>\markdownOptionHelperScriptFileName</code>       | 42, 42–44, 234, 235                  |
| <code>\markdownOptionHybrid</code>                     | 44, 94                               |
| <code>\markdownOptionInputTempFileName</code>          | 42, 231, 232                         |
| <code>\markdownOptionOutputDir</code>                  | 43                                   |
| <code>\markdownOptionOutputTempFileName</code>         | 42, 43, 235                          |
| <code>\markdownOptionSmartEllipses</code>              | 94                                   |
| <code>\markdownOptionStripPercentSigns</code>          | 45, 230, 231                         |
| <code>\markdownOutputFileStream</code>                 | 230                                  |
| <code>\markdownPrepare</code>                          | 229                                  |
| <code>\markdownPrepareLuaOptions</code>                | 227                                  |
| <code>\markdownReadAndConvert</code>                   | 79, 230, 238, 262                    |
| <code>\markdownReadAndConvertProcessLine</code>        | 231, 232                             |
| <code>\markdownReadAndConvertStripPercentSigns</code>  | 231                                  |
| <code>\markdownReadAndConvertTab</code>                | 230                                  |
| <code>\markdownRendererAttributeClassName</code>       | 75                                   |
| <code>\markdownRendererAttributeIdentifier</code>      | 75                                   |
| <code>\markdownRendererAttributeKeyValue</code>        | 75                                   |
| <code>\markdownRendererBlockHtmlCommentBegin</code>    | 73                                   |

|                                              |        |
|----------------------------------------------|--------|
| \markdownRendererBlockHtmlCommentEnd         | 73     |
| \markdownRendererBlockQuoteBegin             | 64     |
| \markdownRendererBlockQuoteEnd               | 65     |
| \markdownRendererCite                        | 72, 72 |
| \markdownRendererCodeSpan                    | 52     |
| \markdownRendererCodeSpanPrototype           | 96     |
| \markdownRendererContentBlock                | 53, 53 |
| \markdownRendererContentBlockCode            | 54     |
| \markdownRendererContentBlockOnlineImage     | 53     |
| \markdownRendererDlBegin                     | 61     |
| \markdownRendererDlBeginTight                | 61     |
| \markdownRendererDlDefinitionBegin           | 62     |
| \markdownRendererDlDefinitionEnd             | 63     |
| \markdownRendererDlEnd                       | 63     |
| \markdownRendererDlEndTight                  | 63     |
| \markdownRendererDlItem                      | 62     |
| \markdownRendererDlItemEnd                   | 62     |
| \markdownRendererDocumentBegin               | 47     |
| \markdownRendererDocumentEnd                 | 47     |
| \markdownRendererEllipsis                    | 30, 48 |
| \markdownRendererEmphasis                    | 64, 95 |
| \markdownRendererFancyOlBegin                | 57, 57 |
| \markdownRendererFancyOlBeginTight           | 57     |
| \markdownRendererFancyOlEnd                  | 60     |
| \markdownRendererFancyOlEndTight             | 61     |
| \markdownRendererFancyOlItem                 | 59     |
| \markdownRendererFancyOlItemEnd              | 59     |
| \markdownRendererFancyOlItemWithNumber       | 59     |
| \markdownRendererFootnote                    | 71     |
| \markdownRendererHalfTickedBox               | 46     |
| \markdownRendererHeaderAttributeContextBegin | 76     |
| \markdownRendererHeaderAttributeContextEnd   | 76     |
| \markdownRendererHeadingFive                 | 70     |
| \markdownRendererHeadingFour                 | 70     |
| \markdownRendererHeadingOne                  | 69     |
| \markdownRendererHeadingSix                  | 71     |
| \markdownRendererHeadingThree                | 70     |
| \markdownRendererHeadingTwo                  | 70     |
| \markdownRendererHorizontalRule              | 71     |
| \markdownRendererImage                       | 52     |
| \markdownRendererImagePrototype              | 96     |
| \markdownRendererInlineHtmlComment           | 73     |

|                                          |                  |
|------------------------------------------|------------------|
| \markdownRendererInlineHtmlTag           | 74               |
| \markdownRendererInputBlockHtmlElement   | 74               |
| \markdownRendererInputFencedCode         | 65               |
| \markdownRendererInputVerbatim           | 65               |
| \markdownRendererInterblockSeparator     | 48               |
| \markdownRendererJekyllDataBegin         | 66               |
| \markdownRendererJekyllDataBoolean       | 68               |
| \markdownRendererJekyllDataEmpty         | 69               |
| \markdownRendererJekyllDataEnd           | 66               |
| \markdownRendererJekyllDataMappingBegin  | 66               |
| \markdownRendererJekyllDataMappingEnd    | 67               |
| \markdownRendererJekyllDataNumber        | 68               |
| \markdownRendererJekyllDataSequenceBegin | 67               |
| \markdownRendererJekyllDataSequenceEnd   | 67               |
| \markdownRendererJekyllDataString        | 68               |
| \markdownRendererLineBreak               | 48               |
| \markdownRendererLink                    | 52, 95           |
| \markdownRendererNbsp                    | 49               |
| \markdownRendererOlBegin                 | 56               |
| \markdownRendererOlBeginTight            | 57               |
| \markdownRendererOlEnd                   | 60               |
| \markdownRendererOlEndTight              | 60               |
| \markdownRendererOlItem                  | 31, 58           |
| \markdownRendererOlItemEnd               | 58               |
| \markdownRendererOlItemWithNumber        | 31, 58           |
| \markdownRendererStrikeThrough           | 76               |
| \markdownRendererStrongEmphasis          | 64               |
| \markdownRendererSubscript               | 77               |
| \markdownRendererSuperscript             | 77               |
| \markdownRendererTable                   | 73               |
| \markdownRendererTextCite                | 72               |
| \markdownRendererTickedBox               | 46               |
| \markdownRendererUlBegin                 | 54               |
| \markdownRendererUlBeginTight            | 55               |
| \markdownRendererUlEnd                   | 56               |
| \markdownRendererUlEndTight              | 56               |
| \markdownRendererUlItem                  | 55               |
| \markdownRendererUlItemEnd               | 55               |
| \markdownRendererUntickedBox             | 46               |
| \markdownSetup                           | 83, 83, 238, 242 |
| \markdownSetupSnippet                    | 83, 83           |
| \markdownWarning                         | 79               |



|                                               |                       |
|-----------------------------------------------|-----------------------|
| <code>new</code>                              | 6, 216                |
| <code>os.execute</code>                       | 80, 234               |
| <code>\PackageError</code>                    | 81, 238               |
| <code>\PackageInfo</code>                     | 81, 238               |
| <code>\PackageWarning</code>                  | 81, 238               |
| <code>parsers</code>                          | 160, 171, 172         |
| <code>parsers.commented_line</code>           | 162                   |
| <code>\pdfshellescape</code>                  | 233                   |
| <code>pipeTables</code>                       | 6, 28, 33, 73         |
| <code>preserveTabs</code>                     | 29, 31, 172           |
| <code>print</code>                            | 234, 236              |
| <code>reader</code>                           | 7, 100, 160, 171, 190 |
| <code>reader-&gt;add_special_character</code> | 7, 8, 185             |
| <code>reader-&gt;create_parser</code>         | 172                   |
| <code>reader-&gt;finalize_grammar</code>      | 183                   |
| <code>reader-&gt;insert_pattern</code>        | 7, 8, 183, 187        |
| <code>reader-&gt;normalize_tag</code>         | 172                   |
| <code>reader-&gt;options</code>               | 171                   |
| <code>reader-&gt;parser_functions</code>      | 172                   |
| <code>reader-&gt;parser_functions.name</code> | 172                   |
| <code>reader-&gt;parsers</code>               | 171, 172              |
| <code>reader-&gt;update_rule</code>           | 183, 185, 187         |
| <code>reader-&gt;writer</code>                | 171                   |
| <code>reader.new</code>                       | 171, 171              |
| <code>relativeReferences</code>               | 29                    |
| <code>\setupmarkdown</code>                   | 98, 98                |
| <code>\shellescape</code>                     | 233                   |
| <code>shiftHeadings</code>                    | 6, 30                 |
| <code>slice</code>                            | 6, 26, 30, 150        |
| <code>smartEllipses</code>                    | 30, 48                |
| <code>\startmarkdown</code>                   | 97, 97, 262           |
| <code>startNumber</code>                      | 31, 58, 59            |
| <code>status.shell_escape</code>              | 233                   |
| <code>\stopmarkdown</code>                    | 97, 97, 262           |
| <code>strikeThrough</code>                    | 31, 76, 260           |
| <code>stripIndent</code>                      | 31, 173               |
| <code>subscripts</code>                       | 32, 77                |
| <code>superscripts</code>                     | 32, 77                |
| <code>syntax</code>                           | 184, 187              |

|                            |                              |
|----------------------------|------------------------------|
| tableCaptions              | 6, 33                        |
| taskLists                  | 33, 46, 250                  |
| tex.print                  | 234, 236                     |
| texComments                | 34, 173                      |
| \textcites                 | 251                          |
| tightLists                 | 34, 55–57, 60–63, 243        |
| \title                     | 259                          |
| underscores                | 35                           |
| \url                       | 4                            |
| \usepackage                | 81, 85                       |
| \usetheme                  | 85                           |
| util.cache                 | 101                          |
| util.encode_json_string    | 101                          |
| util.err                   | 101                          |
| util.escaper               | 104                          |
| util.expand_tabs_in_line   | 102                          |
| util.flatten               | 103                          |
| util.intersperse           | 104                          |
| util.lookup_files          | 102                          |
| util.map                   | 104                          |
| util.pathname              | 105                          |
| util.rope_last             | 103                          |
| util.rope_to_string        | 103                          |
| util.table_copy            | 101                          |
| util.walk                  | 102, 103                     |
| \VerbatimInput             | 4                            |
| walkable_syntax            | 7, 15, 19, 183, 183, 185–187 |
| writer                     | 100, 100, 149, 190           |
| writer->active_attributes  | 156                          |
| writer->active_headings    | 156                          |
| writer->block_html_comment | 154                          |
| writer->block_html_element | 155                          |
| writer->blockquote         | 155                          |
| writer->bulletitem         | 153                          |
| writer->bulletlist         | 153                          |
| writer->citation           | 191                          |
| writer->citations          | 191                          |
| writer->code               | 152                          |
| writer->codeFence          | 199                          |
| writer->contentblock       | 195                          |

|                                 |               |
|---------------------------------|---------------|
| writer->defer_call              | 159, 159      |
| writer->definitionlist          | 198           |
| writer->document                | 156           |
| writer->ellipsis                | 151           |
| writer->emphasis                | 155           |
| writer->escape                  | 152, 152      |
| writer->escape_minimal          | 152, 152, 191 |
| writer->escape_uri              | 152, 152      |
| writer->escaped_chars           | 152, 152      |
| writer->escaped_minimal_strings | 151, 152      |
| writer->escaped_uri_chars       | 151, 152      |
| writer->fancyitem               | 213           |
| writer->fancylist               | 213           |
| writer->get_state               | 159           |
| writer->heading                 | 156           |
| writer->hrule                   | 151           |
| writer->hybrid                  | 191           |
| writer->image                   | 153           |
| writer->inline_html_comment     | 154           |
| writer->inline_html_tag         | 155           |
| writer->interblocksep           | 151           |
| writer->is_writing              | 150, 150      |
| writer->jekyllData              | 205           |
| writer->linebreak               | 151           |
| writer->link                    | 153           |
| writer->nbsp                    | 150           |
| writer->note                    | 202           |
| writer->options                 | 150           |
| writer->ordereditem             | 154           |
| writer->orderedlist             | 153           |
| writer->pack                    | 151, 189      |
| writer->paragraph               | 151           |
| writer->plain                   | 151           |
| writer->set_state               | 159           |
| writer->slice_begin             | 150           |
| writer->slice_end               | 150           |
| writer->space                   | 150           |
| writer->strike_through          | 211           |
| writer->string                  | 152           |
| writer->strong                  | 155           |
| writer->subscript               | 212           |
| writer->suffix                  | 150           |

|                     |               |
|---------------------|---------------|
| writer->superscript | 211           |
| writer->table       | 209           |
| writer->textbox     | 155           |
| writer->uri         | 152           |
| writer->verbatim    | 156           |
| writer.new          | 149, 149, 150 |
| \writestatus        | 97            |