

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.18.0-0-gd8ae860
2022-10-30

Contents

1	Introduction	1	3	Implementation	106
1.1	Requirements	2	3.1	Lua Implementation	106
1.2	Feedback	5	3.2	Plain T _E X Implementation	228
1.3	Acknowledgements	5	3.3	L ^A T _E X Implementation	245
2	Interfaces	6	3.4	ConT _E Xt Implementation	270
2.1	Lua Interface	6		References	276
2.2	Plain T _E X Interface	42		Index	277
2.3	L ^A T _E X Interface	86			
2.4	ConT _E Xt Interface	102			

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	38
3	A sequence diagram of typesetting a document using the Lua CLI	38
4	Various formats of mathematical formulae	93
5	The banner of the Markdown package	94
6	A pushdown automaton that recognizes T _E X comments	169

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 Lua_{TEX} engine:

LPeg \geq 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg \geq 0.10 is included in Lua_{TEX} \geq 0.72.0 (T_EXLive \geq 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 note tags to the lower case. Selene Unicode is included in all releases of Lua_{TEX} (T_EXLive \geq 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 \geq 5.3, we will use the built-in support for Unicode.

```
17   if not ran_ok then
```

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

```

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/luatinyyaml 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).

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 ≥ 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 $2006 \leq \text{T}_{\text{E}}\text{X Live} \leq 2019$. Since $\text{T}_{\text{E}}\text{X Live} \geq 2020$, 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.

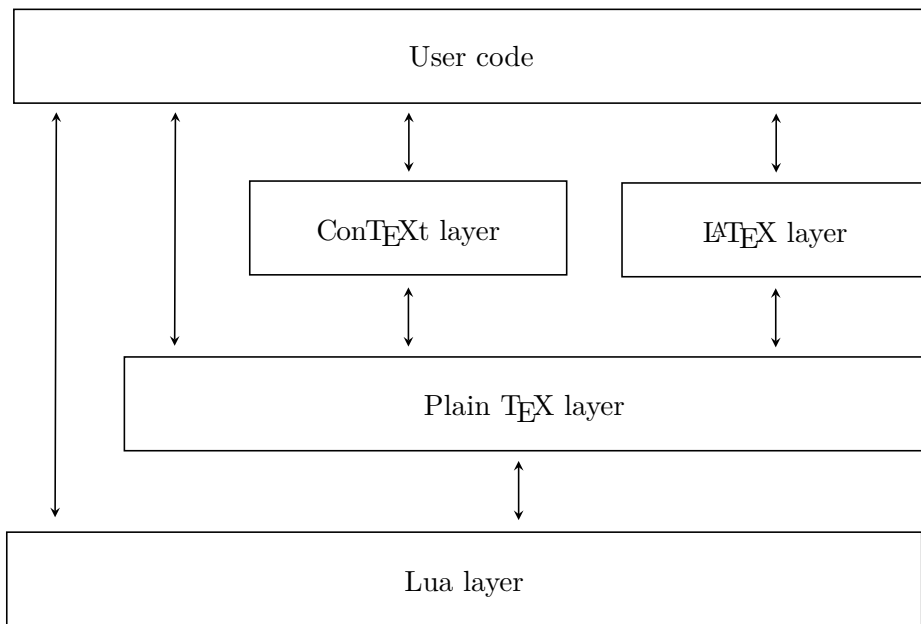


Figure 1: A block diagram of the Markdown package

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     "ThematicBreak",
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_gput_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:nmnn
80   {
81     \seq_gput_right:cn
82     { g_@@_ #1 _options_seq }
83     { #2 }
84     \prop_gput:cnn
85     { g_@@_ #1 _option_types_prop }
86     { #2 }
87     { #3 }
88     \prop_gput: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:nmnn
```

```

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_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
148 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
149 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
150 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
151 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
152 \tl_const:Nn \c_@@_option_type_number_tl { number }
153 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
154 \tl_const:Nn \c_@@_option_type_path_tl { path }
155 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
156 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
157 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
158 \tl_const:Nn \c_@@_option_type_string_tl { string }
159 \seq_gput_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 }
290 \seq_new:N \g_@@_cases_seq
291 \cs_new:Nn \@@_with_various_cases:nn
292 {
293   \seq_clear:N
294   \l_tmpa_seq
295   \seq_map_inline:Nn
296   \g_@@_cases_seq
297   {
298     \tl_set:Nn
299     \l_tmpa_tl
300     { #1 }
301     \use:c { ##1 }
302     \l_tmpa_tl
303     \seq_put_right:NV
304     \l_tmpa_seq
305     \l_tmpa_tl
306   }
307   \seq_map_inline:Nn
308   \l_tmpa_seq
309   { #2 }
310 }
311 \cs_new:Nn \@@_camel_case:N
312 {
313   \regex_replace_all:nnN
314   { _ ([a-z]) }
315   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
316   #1
317   \tl_set:Nx
318   #1
319   { #1 }
320 }
321 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
322 \cs_new:Nn \@@_snake_case:N
323 {
324   \regex_replace_all:nnN
325   { ([a-z])([A-Z]) }
326   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
327   #1
328   \tl_set:Nx
329   #1
330   { #1 }
331 }

```

```
332 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }
```

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.

```
333 \@@_add_lua_option:nnn
334   { cacheDir }
335   { path }
336   { \markdownOptionOutputDir / _markdown_\jobname }
337 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.

```
338 \@@_add_lua_option:nnn
339   { contentBlocksLanguageMap }
340   { path }
341   { markdown-languages.json }
342 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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```

343 \@@_add_lua_option:nnn
344   { debugExtensionsFileName }
345   { path }
346   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
347 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

`frozenCacheFileName`=*<path>* 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.

```

348 \@@_add_lua_option:nnn
349   { frozenCacheFileName }
350   { path }
351   { \markdownOptionCacheDir / frozenCache.tex }
352 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.5 Parser Options

`blankBeforeBlockquote`=`true`, `false` default: `false`

true Require a blank line between a paragraph and the following blockquote.

false Do not require a blank line between a paragraph and the following blockquote.

```

353 \@@_add_lua_option:nnn
354   { blankBeforeBlockquote }
355   { boolean }
356   { false }
357 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.

```
358 \@@_add_lua_option:nnn
359 { blankBeforeCodeFence }
360 { boolean }
361 { false }

362 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.

```
363 \@@_add_lua_option:nnn
364 { blankBeforeHeading }
365 { boolean }
366 { false }

367 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.

```
368 \@@_add_lua_option:nnn
369 { breakableBlockquotes }
370 { boolean }
371 { false }

372 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.

```
373 \@@_add_lua_option:nnn
374 { citationNbsps }
375 { boolean }
376 { true }

377 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.

```
378 \@@_add_lua_option:nnn
379 { citations }
380 { boolean }
381 { false }

382 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.``
```

```
383 \@@_add_lua_option:nnn  
384 { codeSpans }  
385 { boolean }  
386 { true }  
  
387 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.

```
388 \@@_add_lua_option:nnn  
389 { contentBlocks }  
390 { boolean }  
391 { false }  
  
392 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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) 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.

```
393 \@@_add_lua_option:nnn
394   { debugExtensions }
395   { boolean }
396   { false }
397 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.

```
398 \@@_add_lua_option:nnn
399   { definitionLists }
400   { boolean }
401   { false }
402 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.

```
403 \@@_add_lua_option:nnn
404   { eagerCache }
405   { boolean }
406   { true }

407 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 = 2,
  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
  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:

```

408 metadata.user_extension_api_version = 2
409 metadata.grammar_version = 2

```

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>`.

```

410 \cs_generate_variant:Nn
411   \@_add_lua_option:nnn
412   { nnV }
413 \@_add_lua_option:nnV
414   { extensions }
415   { clist }
416   \c_empty_clist
417 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}
```

```
418 \@@_add_lua_option:nnn
419   { expectJekyllData }
420   { boolean }
421   { false }

422 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.

```
423 \@@_add_lua_option:nnn
424   { fancyLists }
425   { boolean }
426   { false }

427 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.

```
428 \@@_add_lua_option:nnn
429 { fencedCode }
430 { boolean }
431 { false }
432 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  $\text{\TeX}$  document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain  $\text{\TeX}$  option. 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.

```
433 \@@_add_lua_option:nnn
434 { finalizeCache }
435 { boolean }
436 { false }
437 defaultOptions.finalizeCache = false
```

`notes=true, false` default: `false`

`true` Enable the Pandoc note syntax extension:

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

[^1]: Here is the note.

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

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

```
{ some.code }
```

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

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

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

The footnotes option has been deprecated and will be removed in Markdown 3.0.0.

```
438 \@@_add_lua_option:nnn
439 { footnotes }
440 { boolean }
441 { false }
442 \@@_add_lua_option:nnn
443 { notes }
444 { boolean }
445 { false }
446 defaultOptions.footnotes = false
447 defaultOptions.notes = false
```

**frozenCacheCounter**=*<number>* 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<number>` that will typeset markdown document number *<number>*.

```
448 \@@_add_lua_option:nnn
449 { frozenCacheCounter }
450 { counter }
451 { 0 }
452 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.

```
453 \@@_add_lua_option:nnn
454 { hardLineBreaks }
455 { boolean }
456 { false }
457 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.

```
458 \@@_add_lua_option:nnn
459 { hashEnumerators }
460 { boolean }
461 { false }
462 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.

```

463 \@@_add_lua_option:nnn
464 { headerAttributes }
465 { boolean }
466 { false }

467 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.

```

468 \@@_add_lua_option:nnn
469 { html }
470 { boolean }
471 { false }

472 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.

```

473 \@@_add_lua_option:nnn
474 { hybrid }
475 { boolean }
476 { false }

477 defaultOptions.hybrid = false

```

`inlineNotes=true, false`

default: false

`true` Enable the Pandoc inline note syntax extension:

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

`false` Disable the Pandoc inline note syntax extension.

The `inlineFootnotes` option has been deprecated and will be removed in Markdown 3.0.0.

```
478 \@@_add_lua_option:nnm
479 { inlineFootnotes }
480 { boolean }
481 { false }
482 \@@_add_lua_option:nnm
483 { inlineNotes }
484 { boolean }
485 { false }

486 defaultOptions.inlineFootnotes = false
487 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`

default: false

`true` Enable the Pandoc `yml_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 `yml_metadata_block` syntax extension for entering metadata in YAML.

```

488 \@@_add_lua_option:nnn
489 { jekyllData }
490 { boolean }
491 { false }

492 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.

```

493 \@@_add_lua_option:nnn
494 { pipeTables }
495 { boolean }
496 { false }

497 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.

```

498 \@@_add_lua_option:nnn
499 { preserveTabs }
500 { boolean }
501 { false }

502 defaultOptions.preserveTabs = false

```

`rawAttribute=true, false`

default: false

`true` Enable the Pandoc raw attribute syntax extension:

```
`${H_2 O}`{=tex} is a liquid.
```

To enable raw blocks, the `fencedCode` option must also be enabled:

```
Here is a mathematical formula:
``` {=tex}
\[distance[i] =
  \begin{dcases}
    a & b \\
    c & d
  \end{dcases}
\]
```

The `rawAttribute` option is a good alternative to the `hybrid` option. Unlike the `hybrid` option, which affects the entire document, the `rawAttribute` option allows you to isolate the parts of your documents that use TeX:

`false` Disable the Pandoc raw attribute syntax extension.

```
503 \@_add_lua_option:nnn
504   { rawAttribute }
505   { boolean }
506   { false }
507 defaultOptions.rawAttribute = true
```

`relativeReferences=true, false`

default: false

`true` Enable relative references⁶ 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!
```

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

`false` Disable relative references in autolinks.

```
508 \@@_add_lua_option:nnn
509   { relativeReferences }
510   { boolean }
511   { false }

512 defaultOptions.relativeReferences = false
```

`shiftHeadings=<shift amount>` default: 0

All headings will be shifted by *<shift amount>*, 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 *<shift amount>* is positive, and to level 1, when *<shift amount>* is negative.

```
513 \@@_add_lua_option:nnn
514   { shiftHeadings }
515   { number }
516   { 0 }

517 defaultOptions.shiftHeadings = 0
```

`slice=<the beginning and the end of a slice>` default: `^ $`

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

Specifying only a single selector, *<identifier>*, is equivalent to specifying the two selectors *<identifier>* *<identifier>*, which is equivalent to `^<identifier>` `$<identifier>`, i.e. the entire section with the HTML attribute `#<identifier>` will be selected.

```
518 \@@_add_lua_option:nnn
519   { slice }
520   { slice }
521   { ^~$ }

522 defaultOptions.slice = "^ $"
```


`smartEllipses=true, false` default: false

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

`false` Preserve all ellipses in the input.

```
523 \@@_add_lua_option:nnn
524 { smartEllipses }
525 { boolean }
526 { false }
527 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 `\markdownRendererOListItemWithNumber` \TeX macro.

`false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOListItem` \TeX macro.

```
528 \@@_add_lua_option:nnn
529 { startNumber }
530 { boolean }
531 { true }
532 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.

```
533 \@@_add_lua_option:nnn
534 { strikeThrough }
535 { boolean }
536 { false }
537 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.

```
538 \@@_add_lua_option:nnn
539   { stripIndent }
540   { boolean }
541   { false }
542 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: false

true Enable the Pandoc subscript syntax extension:

```
H~2~0 is a liquid.
```

false Disable the Pandoc subscript syntax extension.

```
543 \@@_add_lua_option:nnn
544   { subscripts }
545   { boolean }
546   { false }
547 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.

```

548 \@@_add_lua_option:nnn
549   { superscripts }
550   { boolean }
551   { false }

552 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.

```

553 \@@_add_lua_option:nnn
554   { tableCaptions }
555   { boolean }
556   { false }

557 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.

```

558 \@@_add_lua_option:nnn
559   { taskLists }
560   { boolean }
561   { false }

562 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.

```
563 \@@_add_lua_option:nnn
564   { texComments }
565   { boolean }
566   { false }
567 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.

```
568 \@@_add_lua_option:nnn
569   { tightLists }
570   { boolean }
571   { true }
```

```
572 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.

```
573 \@@_add_lua_option:nnn
574 { underscores }
575 { boolean }
576 { true }
577 \ExplSyntaxOff
578 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 $\text{T}_{\text{E}}\text{X}$ layer hands markdown documents to the Lua layer. Lua converts the documents to $\text{T}_{\text{E}}\text{X}$, and hands the converted documents back to plain $\text{T}_{\text{E}}\text{X}$ layer for typesetting, see Figure 2.

This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted $\text{T}_{\text{E}}\text{X}$ documents are cached on the file system, taking up increasing amount of space. Unless the $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ is also provided, see Figure 3.

```
579
580 local HELP_STRING = [[
581 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
582 where OPTIONS are documented in the Lua interface section of the
583 technical Markdown package documentation.
```

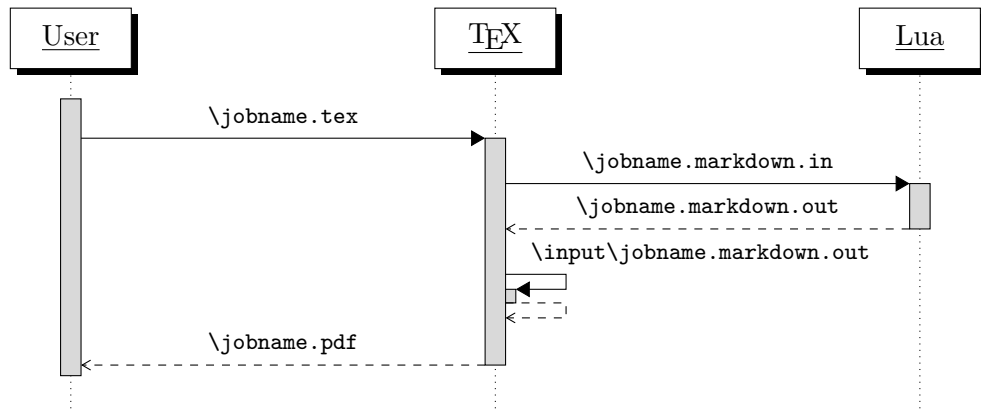


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface

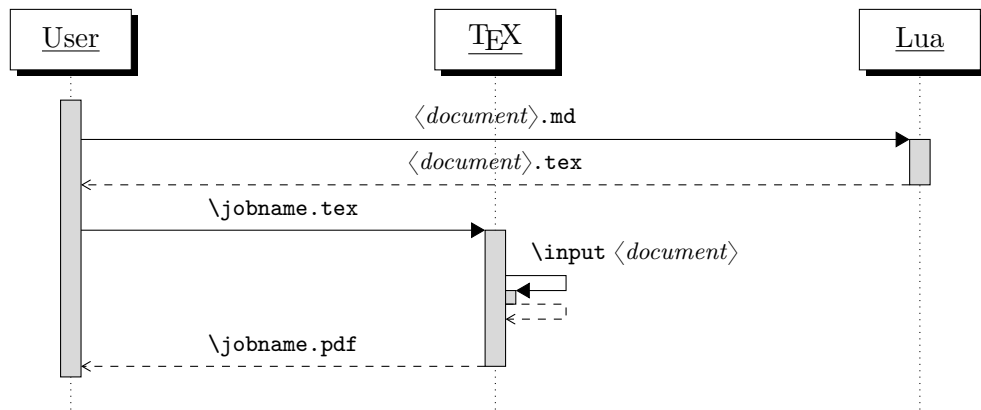


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

584
585 When OUTPUT_FILE is unspecified, the result of the conversion will be
586 written to the standard output. When INPUT_FILE is also unspecified, the
587 result of the conversion will be read from the standard input.
588
589 Report bugs to: witiko@mail.muni.cz
590 Markdown package home page: <https://github.com/witiko/markdown>]]
591
592 local VERSION_STRING = [[
593 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
594
595 Copyright (C) ]] .. table.concat(metadata.copyright,
596                                     "\nCopyright (C) ") .. [[
597
598 License: ]] .. metadata.license
599
600 local function warn(s)
601   io.stderr:write("Warning: " .. s .. "\n") end
602
603 local function error(s)
604   io.stderr:write("Error: " .. s .. "\n")
605   os.exit(1)
606 end

```

To make it easier to copy-and-paste options from Pandoc [4] such as [fancy_lists](#), [header_attributes](#), and [pipe_tables](#), we accept snake_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake_case is faster to read than camelCase.

```

607 local function camel_case(option_name)
608   local cased_option_name = option_name:gsub("_(%l)", function(match)
609     return match:sub(2, 2):upper()
610   end)
611   return cased_option_name
612 end
613
614 local function snake_case(option_name)
615   local cased_option_name = option_name:gsub("%l%u", function(match)
616     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
617   end)
618   return cased_option_name
619 end
620
621 local cases = {camel_case, snake_case}
622 local various_case_options = {}
623 for option_name, _ in pairs(defaultOptions) do
624   for _, case in ipairs(cases) do
625     various_case_options[case(option_name)] = option_name

```

```

626 end
627 end
628
629 local process_options = true
630 local options = {}
631 local input_filename
632 local output_filename
633 for i = 1, #arg do
634   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.

```

635     if arg[i] == "--" then
636       process_options = false
637       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.

```

638     elseif arg[i]:match("=") then
639       local key, value = arg[i]:match("(.)=(.*)")
640       if defaultOptions[key] == nil then
641         key = various_case_options[key]
642       end

```

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

```

643       local default_type = type(defaultOptions[key])
644       if default_type == "boolean" then
645         options[key] = (value == "true")
646       elseif default_type == "number" then
647         options[key] = tonumber(value)
648       elseif default_type == "table" then
649         options[key] = {}
650         for item in value:gmatch("[^,]+") do
651           table.insert(options[key], item)
652         end
653       else
654         if default_type ~= "string" then
655           if default_type == "nil" then
656             warn('Option "' .. key .. '" not recognized.')
657           else
658             warn('Option "' .. key .. '" type not recognized, please file ' ..
659               'a report to the package maintainer.')
660           end
661         end
662         warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..

```



```

662             key .. '"' as a string.')
663         end
664         options[key] = value
665     end
666     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.

```

667     elseif arg[i] == "--help" or arg[i] == "-h" then
668         print(HELP_STRING)
669         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.

```

670     elseif arg[i] == "--version" or arg[i] == "-v" then
671         print(VERSION_STRING)
672         os.exit()
673     end
674 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 \TeX document.

```

675 if input_filename == nil then
676     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 \TeX document that will result from the conversion.

```

677 elseif output_filename == nil then
678     output_filename = arg[i]
679 else
680     error('Unexpected argument: "' .. arg[i] .. "'.')
681 end
682 ::continue::
683 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 \TeX document `hello.tex`. After the Markdown package for our \TeX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

2.2 Plain T_EX Interface

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

```
684 \def\markdownLastModified{((LASTMODIFIED))}%  
685 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown

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

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

```
686 \let\markdownBegin\relax  
687 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T_EX [6, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

Note that you may also not nest the `\markdownBegin` and `\markdownEnd` macros.

The following example plain T_EX code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

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

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

```
688 \let\markdownInput\relax
```

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

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

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

The `\markdownEscape` macro accepts a single parameter with the filename of a T_EX document and executes the T_EX document in the middle of a markdown document fragment. Unlike the `\input` built-in of T_EX, `\markdownEscape` guarantees that the standard catcode regime of your T_EX format will be used.

```
689 \let\markdownEscape\relax
```

2.2.2 Options

The plain \TeX options are represented by \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 \TeX interface.

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

```
690 \ExplSyntaxOn
691 \seq_new:N \g_@@_plain_tex_options_seq
```

To enable the reflection of default plain \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.

```
692 \prop_new:N \g_@@_plain_tex_option_types_prop
693 \prop_new:N \g_@@_default_plain_tex_options_prop
694 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
695 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
696 \cs_new:Nn
697   \@@_add_plain_tex_option:nnn
698   {
699     \@@_add_option:Vnnn
700     \c_@@_option_layer_plain_tex_tl
701     { #1 }
702     { #2 }
703     { #3 }
704   }
```

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 \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 \TeX document without invoking Lua. 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. It defaults to `false`.

```
705 \@@_add_plain_tex_option:nnn
706   { frozenCache }
707   { boolean }
708   { 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_EX document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain T_EX 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_EX in T_EX 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_EX engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
709 \@@_add_plain_tex_option:nnn
710 { helperScriptFileName }
711 { path }
712 { \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.

```
713 \str_new:N
714 \g_luabridge_helper_script_filename_str
715 \tl_gset:Nn
716 \g_luabridge_helper_script_filename_str
717 { \markdownOptionHelperScriptFileName }
```

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

```
718 \@@_add_plain_tex_option:nnn
719 { inputTempFileName }
720 { path }
721 { \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_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
722 \@@_add_plain_tex_option:nnn
723 { outputTempFileName }
724 { path }
725 { \jobname.markdown.out }
```

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

```
726 \str_new:N
727 \g_luabridge_standard_output_filename_str
728 \tl_gset:Nn
729 \g_luabridge_standard_output_filename_str
730 { \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_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
731 \@@_add_plain_tex_option:nnn
732 { errorTempFileName }
733 { path }
734 { \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.

```
735 \str_new:N
736 \g_luabridge_error_output_filename_str
737 \tl_gset:Nn
738 \g_luabridge_error_output_filename_str
739 { \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_EX implementation. The option defaults to `..`.

The path must be set to the same value as the `-output-directory` option of your T_EX 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.

```
740 \@@_add_plain_tex_option:nnn
741 { outputDir }
742 { path }
743 { . }
```

Here, we automatically define plain T_EX macros for the above plain T_EX 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 T_EX 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.

```

744 \cs_new:Nn \@@_plain_tex_define_option_commands:
745   {
746     \seq_map_inline:Nn
747       \g_@@_option_layers_seq
748       {
749         \seq_map_inline:cn
750           { g_@@_ ##1 _options_seq }
751           {
752             \@@_plain_tex_define_option_command:n
753             { ####1 }
754           }
755       }
756   }
757 \cs_new:Nn \@@_plain_tex_define_option_command:n
758   {
759     \@@_get_default_option_value:nN
760     { #1 }
761     \l_tmpa_tl
762     \@@_set_option_value:nV
763     { #1 }
764     \l_tmpa_tl
765   }
766 \cs_new:Nn
767   \@@_set_option_value:nn
768   {
769     \@@_define_option:n
770     { #1 }
771     \@@_get_option_type:nN
772     { #1 }
773     \l_tmpa_tl
774     \str_if_eq:NNTF
775     \c_@@_option_type_counter_tl
776     \l_tmpa_tl
777     {
778       \@@_option_tl_to_csname:nN
779       { #1 }
780       \l_tmpa_tl
781       \int_gset:cn
782       { \l_tmpa_tl }
783       { #2 }
784     }
785     {
786       \@@_option_tl_to_csname:nN
787       { #1 }

```

```

788         \l_tmpa_tl
789         \cs_set:cpn
790         { \l_tmpa_tl }
791         { #2 }
792     }
793 }
794 \cs_generate_variant:Nn
795 \@@_set_option_value:nn
796 { nV }
797 \cs_new:Nn
798 \@@_define_option:n
799 {
800     \@@_option_tl_to_csname:nN
801     { #1 }
802     \l_tmpa_tl
803     \cs_if_free:cT
804     { \l_tmpa_tl }
805     {
806         \@@_get_option_type:nN
807         { #1 }
808         \l_tmpb_tl
809         \str_if_eq:NNT
810         \c_@@_option_type_counter_tl
811         \l_tmpb_tl
812         {
813             \@@_option_tl_to_csname:nN
814             { #1 }
815             \l_tmpa_tl
816             \int_new:c
817             { \l_tmpa_tl }
818         }
819     }
820 }
821 \@@_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 \TeX package documentation using the Doc \LaTeX package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

822 \seq_gput_right:Nn
823 \g_@@_plain_tex_options_seq
824 { stripPercentSigns }
825 \prop_gput:Nnn
826 \g_@@_plain_tex_option_types_prop

```



```

827 { stripPercentSigns }
828 { boolean }
829 \prop_gput:Nnx
830 \g_@@_default_plain_tex_options_prop
831 { stripPercentSigns }
832 { false }
833 \ExplSyntaxOff

```

2.2.3 Token Renderers

The following T_EX 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.

```

834 \ExplSyntaxOn
835 \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.

```

836 \prop_new:N \g_@@_renderer_arities_prop
837 \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.

```

838 \def\markdownRendererTickedBox{%
839 \markdownRendererTickedBoxPrototype}%
840 \ExplSyntaxOn
841 \seq_gput_right:Nn
842 \g_@@_renderers_seq
843 { tickedBox }
844 \prop_gput:Nnn
845 \g_@@_renderer_arities_prop
846 { tickedBox }
847 { 0 }
848 \ExplSyntaxOff
849 \def\markdownRendererHalfTickedBox{%
850 \markdownRendererHalfTickedBoxPrototype}%
851 \ExplSyntaxOn
852 \seq_gput_right:Nn

```

```

853 \g_@@_renderers_seq
854 { halfTickedBox }
855 \prop_gput:Nnn
856 \g_@@_renderer_arities_prop
857 { halfTickedBox }
858 { 0 }
859 \ExplSyntaxOff
860 \def\markdownRendererUntickedBox{%
861 \markdownRendererUntickedBoxPrototype}%
862 \ExplSyntaxOn
863 \seq_gput_right:Nn
864 \g_@@_renderers_seq
865 { untickedBox }
866 \prop_gput:Nnn
867 \g_@@_renderer_arities_prop
868 { untickedBox }
869 { 0 }
870 \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 \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.

```

871 \def\markdownRendererDocumentBegin{%
872 \markdownRendererDocumentBeginPrototype}%
873 \ExplSyntaxOn
874 \seq_gput_right:Nn
875 \g_@@_renderers_seq
876 { documentBegin }
877 \prop_gput:Nnn
878 \g_@@_renderer_arities_prop
879 { documentBegin }
880 { 0 }
881 \ExplSyntaxOff
882 \def\markdownRendererDocumentEnd{%
883 \markdownRendererDocumentEndPrototype}%
884 \ExplSyntaxOn
885 \seq_gput_right:Nn
886 \g_@@_renderers_seq
887 { documentEnd }
888 \prop_gput:Nnn
889 \g_@@_renderer_arities_prop

```

```

890 { documentEnd }
891 { 0 }
892 \ExplSyntaxOff

```

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

```

893 \def\markdownRendererInterblockSeparator{%
894 \markdownRendererInterblockSeparatorPrototype}%
895 \ExplSyntaxOn
896 \seq_gput_right:Nn
897 \g_@@_renderers_seq
898 { interblockSeparator }
899 \prop_gput:Nnn
900 \g_@@_renderer_arities_prop
901 { interblockSeparator }
902 { 0 }
903 \ExplSyntaxOff

```

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

```

904 \def\markdownRendererLineBreak{%
905 \markdownRendererLineBreakPrototype}%
906 \ExplSyntaxOn
907 \seq_gput_right:Nn
908 \g_@@_renderers_seq
909 { lineBreak }
910 \prop_gput:Nnn
911 \g_@@_renderer_arities_prop
912 { lineBreak }
913 { 0 }
914 \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.

```

915 \def\markdownRendererEllipsis{%
916 \markdownRendererEllipsisPrototype}%
917 \ExplSyntaxOn
918 \seq_gput_right:Nn
919 \g_@@_renderers_seq
920 { ellipsis }
921 \prop_gput:Nnn
922 \g_@@_renderer_arities_prop

```

```

923 { ellipsis }
924 { 0 }
925 \ExplSyntaxOff

```

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

```

926 \def\markdownRendererNbsp{%
927   \markdownRendererNbspPrototype}%
928 \ExplSyntaxOn
929 \seq_gput_right:Nn
930   \g_@@_renderers_seq
931   { nbsp }
932 \prop_gput:Nnn
933   \g_@@_renderer_arities_prop
934   { nbsp }
935   { 0 }
936 \ExplSyntaxOff

```

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

```

937 \def\markdownRendererLeftBrace{%
938   \markdownRendererLeftBracePrototype}%
939 \ExplSyntaxOn
940 \seq_gput_right:Nn
941   \g_@@_renderers_seq
942   { leftBrace }
943 \prop_gput:Nnn
944   \g_@@_renderer_arities_prop
945   { leftBrace }
946   { 0 }
947 \ExplSyntaxOff
948 \def\markdownRendererRightBrace{%
949   \markdownRendererRightBracePrototype}%
950 \ExplSyntaxOn
951 \seq_gput_right:Nn
952   \g_@@_renderers_seq
953   { rightBrace }
954 \prop_gput:Nnn
955   \g_@@_renderer_arities_prop
956   { rightBrace }
957   { 0 }
958 \ExplSyntaxOff
959 \def\markdownRendererDollarSign{%
960   \markdownRendererDollarSignPrototype}%

```

```

961 \ExplSyntaxOn
962 \seq_gput_right:Nn
963   \g_@@_renderers_seq
964   { dollarSign }
965 \prop_gput:Nnn
966   \g_@@_renderer_arities_prop
967   { dollarSign }
968   { 0 }
969 \ExplSyntaxOff
970 \def\markdownRendererPercentSign{%
971   \markdownRendererPercentSignPrototype}%
972 \ExplSyntaxOn
973 \seq_gput_right:Nn
974   \g_@@_renderers_seq
975   { percentSign }
976 \prop_gput:Nnn
977   \g_@@_renderer_arities_prop
978   { percentSign }
979   { 0 }
980 \ExplSyntaxOff
981 \def\markdownRendererAmpersand{%
982   \markdownRendererAmpersandPrototype}%
983 \ExplSyntaxOn
984 \seq_gput_right:Nn
985   \g_@@_renderers_seq
986   { ampersand }
987 \prop_gput:Nnn
988   \g_@@_renderer_arities_prop
989   { ampersand }
990   { 0 }
991 \ExplSyntaxOff
992 \def\markdownRendererUnderscore{%
993   \markdownRendererUnderscorePrototype}%
994 \ExplSyntaxOn
995 \seq_gput_right:Nn
996   \g_@@_renderers_seq
997   { underscore }
998 \prop_gput:Nnn
999   \g_@@_renderer_arities_prop
1000  { underscore }
1001  { 0 }
1002 \ExplSyntaxOff
1003 \def\markdownRendererHash{%
1004   \markdownRendererHashPrototype}%
1005 \ExplSyntaxOn
1006 \seq_gput_right:Nn
1007   \g_@@_renderers_seq

```

```

1008 { hash }
1009 \prop_gput:Nnn
1010 \g_@@_renderer_arities_prop
1011 { hash }
1012 { 0 }
1013 \ExplSyntaxOff
1014 \def\markdownRendererCircumflex{%
1015 \markdownRendererCircumflexPrototype}%
1016 \ExplSyntaxOn
1017 \seq_gput_right:Nn
1018 \g_@@_renderers_seq
1019 { circumflex }
1020 \prop_gput:Nnn
1021 \g_@@_renderer_arities_prop
1022 { circumflex }
1023 { 0 }
1024 \ExplSyntaxOff
1025 \def\markdownRendererBackslash{%
1026 \markdownRendererBackslashPrototype}%
1027 \ExplSyntaxOn
1028 \seq_gput_right:Nn
1029 \g_@@_renderers_seq
1030 { backslash }
1031 \prop_gput:Nnn
1032 \g_@@_renderer_arities_prop
1033 { backslash }
1034 { 0 }
1035 \ExplSyntaxOff
1036 \def\markdownRendererTilde{%
1037 \markdownRendererTildePrototype}%
1038 \ExplSyntaxOn
1039 \seq_gput_right:Nn
1040 \g_@@_renderers_seq
1041 { tilde }
1042 \prop_gput:Nnn
1043 \g_@@_renderer_arities_prop
1044 { tilde }
1045 { 0 }
1046 \ExplSyntaxOff
1047 \def\markdownRendererPipe{%
1048 \markdownRendererPipePrototype}%
1049 \ExplSyntaxOn
1050 \seq_gput_right:Nn
1051 \g_@@_renderers_seq
1052 { pipe }
1053 \prop_gput:Nnn
1054 \g_@@_renderer_arities_prop

```

```

1055 { pipe }
1056 { 0 }
1057 \ExplSyntaxOff

```

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

```

1058 \def\markdownRendererCodeSpan{%
1059 \markdownRendererCodeSpanPrototype}%
1060 \ExplSyntaxOn
1061 \seq_gput_right:Nn
1062 \g_@@_renderers_seq
1063 { codeSpan }
1064 \prop_gput:Nnn
1065 \g_@@_renderer_arities_prop
1066 { codeSpan }
1067 { 1 }
1068 \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.

```

1069 \def\markdownRendererLink{%
1070 \markdownRendererLinkPrototype}%
1071 \ExplSyntaxOn
1072 \seq_gput_right:Nn
1073 \g_@@_renderers_seq
1074 { link }
1075 \prop_gput:Nnn
1076 \g_@@_renderer_arities_prop
1077 { link }
1078 { 4 }
1079 \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.

```

1080 \def\markdownRendererImage{%
1081 \markdownRendererImagePrototype}%
1082 \ExplSyntaxOn
1083 \seq_gput_right:Nn
1084 \g_@@_renderers_seq
1085 { image }

```

```

1086 \prop_gput:Nnn
1087   \g_@@_renderer_arities_prop
1088   { image }
1089   { 4 }
1090 \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.

```

1091 \def\markdownRendererContentBlock{%
1092   \markdownRendererContentBlockPrototype}%
1093 \ExplSyntaxOn
1094 \seq_gput_right:Nn
1095   \g_@@_renderers_seq
1096   { contentBlock }
1097 \prop_gput:Nnn
1098   \g_@@_renderer_arities_prop
1099   { contentBlock }
1100   { 4 }
1101 \ExplSyntaxOff

```

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

```

1102 \def\markdownRendererContentBlockOnlineImage{%
1103   \markdownRendererContentBlockOnlineImagePrototype}%
1104 \ExplSyntaxOn
1105 \seq_gput_right:Nn
1106   \g_@@_renderers_seq
1107   { contentBlockOnlineImage }
1108 \prop_gput:Nnn
1109   \g_@@_renderer_arities_prop
1110   { contentBlockOnlineImage }
1111   { 4 }
1112 \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`⁷ 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 .

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

The macro receives five arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local \TeX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```

1113 \def\markdownRendererContentBlockCode{%
1114   \markdownRendererContentBlockCodePrototype}%
1115 \ExplSyntaxOn
1116 \seq_gput_right:Nn
1117   \g_@@_renderers_seq
1118   { contentBlockCode }
1119 \prop_gput:Nnn
1120   \g_@@_renderer_arities_prop
1121   { contentBlockCode }
1122   { 5 }
1123 \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.

```

1124 \def\markdownRendererUllBegin{%
1125   \markdownRendererUllBeginPrototype}%
1126 \ExplSyntaxOn
1127 \seq_gput_right:Nn
1128   \g_@@_renderers_seq
1129   { ullBegin }
1130 \prop_gput:Nnn
1131   \g_@@_renderer_arities_prop
1132   { ullBegin }
1133   { 0 }
1134 \ExplSyntaxOff

```

The `\markdownRendererUllBeginTight` 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.

```

1135 \def\markdownRendererUllBeginTight{%
1136   \markdownRendererUllBeginTightPrototype}%
1137 \ExplSyntaxOn
1138 \seq_gput_right:Nn

```

```

1139 \g_@@_renderers_seq
1140 { ulBeginTight }
1141 \prop_gput:Nnn
1142 \g_@@_renderer_arities_prop
1143 { ulBeginTight }
1144 { 0 }
1145 \ExplSyntaxOff

```

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

```

1146 \def\markdownRendererUListItem{%
1147 \markdownRendererUListItemPrototype}%
1148 \ExplSyntaxOn
1149 \seq_gput_right:Nn
1150 \g_@@_renderers_seq
1151 { ulItem }
1152 \prop_gput:Nnn
1153 \g_@@_renderer_arities_prop
1154 { ulItem }
1155 { 0 }
1156 \ExplSyntaxOff

```

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

```

1157 \def\markdownRendererUItemEnd{%
1158 \markdownRendererUItemEndPrototype}%
1159 \ExplSyntaxOn
1160 \seq_gput_right:Nn
1161 \g_@@_renderers_seq
1162 { ulItemEnd }
1163 \prop_gput:Nnn
1164 \g_@@_renderer_arities_prop
1165 { ulItemEnd }
1166 { 0 }
1167 \ExplSyntaxOff

```

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

```

1168 \def\markdownRendererUListEnd{%
1169 \markdownRendererUListEndPrototype}%
1170 \ExplSyntaxOn
1171 \seq_gput_right:Nn
1172 \g_@@_renderers_seq
1173 { ulEnd }
1174 \prop_gput:Nnn

```

```

1175 \g_@@_renderer_arities_prop
1176 { ulEnd }
1177 { 0 }
1178 \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.

```

1179 \def\markdownRendererUEndTight{%
1180 \markdownRendererUEndTightPrototype}%
1181 \ExplSyntaxOn
1182 \seq_gput_right:Nn
1183 \g_@@_renderers_seq
1184 { ulEndTight }
1185 \prop_gput:Nnn
1186 \g_@@_renderer_arities_prop
1187 { ulEndTight }
1188 { 0 }
1189 \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.

```

1190 \def\markdownRendererOlBegin{%
1191 \markdownRendererOlBeginPrototype}%
1192 \ExplSyntaxOn
1193 \seq_gput_right:Nn
1194 \g_@@_renderers_seq
1195 { olBegin }
1196 \prop_gput:Nnn
1197 \g_@@_renderer_arities_prop
1198 { olBegin }
1199 { 0 }
1200 \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.

```

1201 \def\markdownRendererOlBeginTight{%
1202 \markdownRendererOlBeginTightPrototype}%
1203 \ExplSyntaxOn
1204 \seq_gput_right:Nn

```

```

1205 \g_@@_renderers_seq
1206 { olBeginTight }
1207 \prop_gput:Nnn
1208 \g_@@_renderers_arity_prop
1209 { olBeginTight }
1210 { 0 }
1211 \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`).

```

1212 \def\markdownRendererFancyOlBegin{%
1213 \markdownRendererFancyOlBeginPrototype}%
1214 \ExplSyntaxOn
1215 \seq_gput_right:Nn
1216 \g_@@_renderers_seq
1217 { fancyOlBegin }
1218 \prop_gput:Nnn
1219 \g_@@_renderers_arity_prop
1220 { fancyOlBegin }
1221 { 2 }
1222 \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.

```

1223 \def\markdownRendererFancyOlBeginTight{%
1224 \markdownRendererFancyOlBeginTightPrototype}%
1225 \ExplSyntaxOn
1226 \seq_gput_right:Nn
1227 \g_@@_renderers_seq
1228 { fancyOlBeginTight }
1229 \prop_gput:Nnn
1230 \g_@@_renderers_arity_prop
1231 { fancyOlBeginTight }
1232 { 2 }
1233 \ExplSyntaxOff

```

The `\markdownRenderer01Item` 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.

```

1234 \def\markdownRenderer01Item{%
1235   \markdownRenderer01ItemPrototype}%
1236 \ExplSyntaxOn
1237 \seq_gput_right:Nn
1238   \g_@@_renderers_seq
1239   { olItem }
1240 \prop_gput:Nnn
1241   \g_@@_renderer_arities_prop
1242   { olItem }
1243   { 0 }
1244 \ExplSyntaxOff

```

The `\markdownRenderer01ItemEnd` 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.

```

1245 \def\markdownRenderer01ItemEnd{%
1246   \markdownRenderer01ItemEndPrototype}%
1247 \ExplSyntaxOn
1248 \seq_gput_right:Nn
1249   \g_@@_renderers_seq
1250   { olItemEnd }
1251 \prop_gput:Nnn
1252   \g_@@_renderer_arities_prop
1253   { olItemEnd }
1254   { 0 }
1255 \ExplSyntaxOff

```

The `\markdownRenderer01ItemWithNumber` 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.

```

1256 \def\markdownRenderer01ItemWithNumber{%
1257   \markdownRenderer01ItemWithNumberPrototype}%
1258 \ExplSyntaxOn
1259 \seq_gput_right:Nn
1260   \g_@@_renderers_seq
1261   { olItemWithNumber }
1262 \prop_gput:Nnn
1263   \g_@@_renderer_arities_prop
1264   { olItemWithNumber }
1265   { 1 }
1266 \ExplSyntaxOff

```

The `\markdownRendererFancy01Item` 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.

```

1267 \def\markdownRendererFancy01Item{%
1268   \markdownRendererFancy01ItemPrototype}%
1269 \ExplSyntaxOn
1270 \seq_gput_right:Nn
1271   \g_@@_renderers_seq
1272   { fancy01Item }
1273 \prop_gput:Nnn
1274   \g_@@_renderer_arities_prop
1275   { fancy01Item }
1276   { 0 }
1277 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemEnd` 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.

```

1278 \def\markdownRendererFancy01ItemEnd{%
1279   \markdownRendererFancy01ItemEndPrototype}%
1280 \ExplSyntaxOn
1281 \seq_gput_right:Nn
1282   \g_@@_renderers_seq
1283   { fancy01ItemEnd }
1284 \prop_gput:Nnn
1285   \g_@@_renderer_arities_prop
1286   { fancy01ItemEnd }
1287   { 0 }
1288 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemWithNumber` 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.

```

1289 \def\markdownRendererFancy01ItemWithNumber{%
1290   \markdownRendererFancy01ItemWithNumberPrototype}%
1291 \ExplSyntaxOn
1292 \seq_gput_right:Nn
1293   \g_@@_renderers_seq
1294   { fancy01ItemWithNumber }
1295 \prop_gput:Nnn
1296   \g_@@_renderer_arities_prop
1297   { fancy01ItemWithNumber }
1298   { 1 }
1299 \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.

```

1300 \def\markdownRendererO1End{%
1301   \markdownRendererO1EndPrototype}%
1302 \ExplSyntaxOn
1303 \seq_gput_right:Nn
1304   \g_@@_renderers_seq
1305   { olEnd }
1306 \prop_gput:Nnn
1307   \g_@@_renderer_arities_prop
1308   { olEnd }
1309   { 0 }
1310 \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.

```

1311 \def\markdownRendererO1EndTight{%
1312   \markdownRendererO1EndTightPrototype}%
1313 \ExplSyntaxOn
1314 \seq_gput_right:Nn
1315   \g_@@_renderers_seq
1316   { olEndTight }
1317 \prop_gput:Nnn
1318   \g_@@_renderer_arities_prop
1319   { olEndTight }
1320   { 0 }
1321 \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.

```

1322 \def\markdownRendererFancyO1End{%
1323   \markdownRendererFancyO1EndPrototype}%
1324 \ExplSyntaxOn
1325 \seq_gput_right:Nn
1326   \g_@@_renderers_seq
1327   { fancyO1End }
1328 \prop_gput:Nnn
1329   \g_@@_renderer_arities_prop
1330   { fancyO1End }

```

```

1331 { 0 }
1332 \ExplSyntaxOff

```

The `\markdownRendererFancy01EndTight` 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.

```

1333 \def\markdownRendererFancy01EndTight{%
1334 \markdownRendererFancy01EndTightPrototype}%
1335 \ExplSyntaxOn
1336 \seq_gput_right:Nn
1337 \g_@@_renderers_seq
1338 { fancy01EndTight }
1339 \prop_gput:Nnn
1340 \g_@@_renderer_arities_prop
1341 { fancy01EndTight }
1342 { 0 }
1343 \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.

```

1344 \def\markdownRendererDlBegin{%
1345 \markdownRendererDlBeginPrototype}%
1346 \ExplSyntaxOn
1347 \seq_gput_right:Nn
1348 \g_@@_renderers_seq
1349 { dlBegin }
1350 \prop_gput:Nnn
1351 \g_@@_renderer_arities_prop
1352 { dlBegin }
1353 { 0 }
1354 \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.

```

1355 \def\markdownRendererDlBeginTight{%
1356 \markdownRendererDlBeginTightPrototype}%
1357 \ExplSyntaxOn
1358 \seq_gput_right:Nn
1359 \g_@@_renderers_seq

```



```

1360 { dlBeginTight }
1361 \prop_gput:Nnn
1362 \g_@@_renderer_arities_prop
1363 { dlBeginTight }
1364 { 0 }
1365 \ExplSyntaxOff

```

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

```

1366 \def\markdownRendererDlItem{%
1367 \markdownRendererDlItemPrototype}%
1368 \ExplSyntaxOn
1369 \seq_gput_right:Nn
1370 \g_@@_renderers_seq
1371 { dlItem }
1372 \prop_gput:Nnn
1373 \g_@@_renderer_arities_prop
1374 { dlItem }
1375 { 1 }
1376 \ExplSyntaxOff

```

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

```

1377 \def\markdownRendererDlItemEnd{%
1378 \markdownRendererDlItemEndPrototype}%
1379 \ExplSyntaxOn
1380 \seq_gput_right:Nn
1381 \g_@@_renderers_seq
1382 { dlItemEnd }
1383 \prop_gput:Nnn
1384 \g_@@_renderer_arities_prop
1385 { dlItemEnd }
1386 { 0 }
1387 \ExplSyntaxOff

```

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

```

1388 \def\markdownRendererDlDefinitionBegin{%
1389 \markdownRendererDlDefinitionBeginPrototype}%
1390 \ExplSyntaxOn
1391 \seq_gput_right:Nn
1392 \g_@@_renderers_seq
1393 { dlDefinitionBegin }
1394 \prop_gput:Nnn
1395 \g_@@_renderer_arities_prop
1396 { dlDefinitionBegin }
1397 { 0 }

```

1398 \ExplSyntaxOff

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

```
1399 \def\markdownRendererDlDefinitionEnd{%
1400   \markdownRendererDlDefinitionEndPrototype}%
1401 \ExplSyntaxOn
1402 \seq_gput_right:Nn
1403   \g_@@_renderers_seq
1404   { dlDefinitionEnd }
1405 \prop_gput:Nnn
1406   \g_@@_renderer_arities_prop
1407   { dlDefinitionEnd }
1408   { 0 }
1409 \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.

```
1410 \def\markdownRendererDlEnd{%
1411   \markdownRendererDlEndPrototype}%
1412 \ExplSyntaxOn
1413 \seq_gput_right:Nn
1414   \g_@@_renderers_seq
1415   { dlEnd }
1416 \prop_gput:Nnn
1417   \g_@@_renderer_arities_prop
1418   { dlEnd }
1419   { 0 }
1420 \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.

```
1421 \def\markdownRendererDlEndTight{%
1422   \markdownRendererDlEndTightPrototype}%
1423 \ExplSyntaxOn
1424 \seq_gput_right:Nn
1425   \g_@@_renderers_seq
1426   { dlEndTight }
1427 \prop_gput:Nnn
1428   \g_@@_renderer_arities_prop
1429   { dlEndTight }
1430   { 0 }
1431 \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.

```

1432 \def\markdownRendererEmphasis{%
1433   \markdownRendererEmphasisPrototype}%
1434 \ExplSyntaxOn
1435 \seq_gput_right:Nn
1436   \g_@@_renderers_seq
1437   { emphasis }
1438 \prop_gput:Nnn
1439   \g_@@_renderer_arities_prop
1440   { emphasis }
1441   { 1 }
1442 \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.

```

1443 \def\markdownRendererStrongEmphasis{%
1444   \markdownRendererStrongEmphasisPrototype}%
1445 \ExplSyntaxOn
1446 \seq_gput_right:Nn
1447   \g_@@_renderers_seq
1448   { strongEmphasis }
1449 \prop_gput:Nnn
1450   \g_@@_renderer_arities_prop
1451   { strongEmphasis }
1452   { 1 }
1453 \ExplSyntaxOff

```

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

```

1454 \def\markdownRendererBlockQuoteBegin{%
1455   \markdownRendererBlockQuoteBeginPrototype}%
1456 \ExplSyntaxOn
1457 \seq_gput_right:Nn
1458   \g_@@_renderers_seq
1459   { blockQuoteBegin }
1460 \prop_gput:Nnn
1461   \g_@@_renderer_arities_prop
1462   { blockQuoteBegin }
1463   { 0 }
1464 \ExplSyntaxOff

```

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

```
1465 \def\markdownRendererBlockQuoteEnd{%
1466   \markdownRendererBlockQuoteEndPrototype}%
1467 \ExplSyntaxOn
1468 \seq_gput_right:Nn
1469   \g_@@_renderers_seq
1470   { blockQuoteEnd }
1471 \prop_gput:Nnn
1472   \g_@@_renderer_arities_prop
1473   { blockQuoteEnd }
1474   { 0 }
1475 \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.

```
1476 \def\markdownRendererInputVerbatim{%
1477   \markdownRendererInputVerbatimPrototype}%
1478 \ExplSyntaxOn
1479 \seq_gput_right:Nn
1480   \g_@@_renderers_seq
1481   { inputVerbatim }
1482 \prop_gput:Nnn
1483   \g_@@_renderer_arities_prop
1484   { inputVerbatim }
1485   { 1 }
1486 \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 infosting.

```
1487 \def\markdownRendererInputFencedCode{%
1488   \markdownRendererInputFencedCodePrototype}%
1489 \ExplSyntaxOn
1490 \seq_gput_right:Nn
1491   \g_@@_renderers_seq
1492   { inputFencedCode }
1493 \prop_gput:Nnn
1494   \g_@@_renderer_arities_prop
1495   { inputFencedCode }
1496   { 2 }
1497 \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.

```

1498 \def\markdownRendererJekyllDataBegin{%
1499   \markdownRendererJekyllDataBeginPrototype}%
1500 \ExplSyntaxOn
1501 \seq_gput_right:Nn
1502   \g_@@_renderers_seq
1503   { jekyllDataBegin }
1504 \prop_gput:Nnn
1505   \g_@@_renderer_arities_prop
1506   { jekyllDataBegin }
1507   { 0 }
1508 \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.

```

1509 \def\markdownRendererJekyllDataEnd{%
1510   \markdownRendererJekyllDataEndPrototype}%
1511 \ExplSyntaxOn
1512 \seq_gput_right:Nn
1513   \g_@@_renderers_seq
1514   { jekyllDataEnd }
1515 \prop_gput:Nnn
1516   \g_@@_renderer_arities_prop
1517   { jekyllDataEnd }
1518   { 0 }
1519 \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.

```

1520 \def\markdownRendererJekyllDataMappingBegin{%
1521   \markdownRendererJekyllDataMappingBeginPrototype}%
1522 \ExplSyntaxOn
1523 \seq_gput_right:Nn
1524   \g_@@_renderers_seq
1525   { jekyllDataMappingBegin }
1526 \prop_gput:Nnn
1527   \g_@@_renderer_arities_prop
1528   { jekyllDataMappingBegin }
1529   { 2 }
1530 \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.

```

1531 \def\markdownRendererJekyllDataMappingEnd{%
1532   \markdownRendererJekyllDataMappingEndPrototype}%
1533 \ExplSyntaxOn
1534 \seq_gput_right:Nn
1535   \g_@@_renderers_seq
1536   { jekyllDataMappingEnd }
1537 \prop_gput:Nnn
1538   \g_@@_renderer_arities_prop
1539   { jekyllDataMappingEnd }
1540   { 0 }
1541 \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.

```

1542 \def\markdownRendererJekyllDataSequenceBegin{%
1543   \markdownRendererJekyllDataSequenceBeginPrototype}%
1544 \ExplSyntaxOn
1545 \seq_gput_right:Nn
1546   \g_@@_renderers_seq
1547   { jekyllDataSequenceBegin }
1548 \prop_gput:Nnn
1549   \g_@@_renderer_arities_prop
1550   { jekyllDataSequenceBegin }
1551   { 2 }
1552 \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.

```

1553 \def\markdownRendererJekyllDataSequenceEnd{%
1554   \markdownRendererJekyllDataSequenceEndPrototype}%
1555 \ExplSyntaxOn
1556 \seq_gput_right:Nn
1557   \g_@@_renderers_seq
1558   { jekyllDataSequenceEnd }
1559 \prop_gput:Nnn
1560   \g_@@_renderer_arities_prop
1561   { jekyllDataSequenceEnd }
1562   { 0 }
1563 \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.

```
1564 \def\markdownRendererJekyllDataBoolean{%
1565   \markdownRendererJekyllDataBooleanPrototype}%
1566 \ExplSyntaxOn
1567 \seq_gput_right:Nn
1568   \g_@@_renderers_seq
1569   { jekyllDataBoolean }
1570 \prop_gput:Nnn
1571   \g_@@_renderer_arities_prop
1572   { jekyllDataBoolean }
1573   { 2 }
1574 \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.

```
1575 \def\markdownRendererJekyllDataNumber{%
1576   \markdownRendererJekyllDataNumberPrototype}%
1577 \ExplSyntaxOn
1578 \seq_gput_right:Nn
1579   \g_@@_renderers_seq
1580   { jekyllDataNumber }
1581 \prop_gput:Nnn
1582   \g_@@_renderer_arities_prop
1583   { jekyllDataNumber }
1584   { 2 }
1585 \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.

```
1586 \def\markdownRendererJekyllDataString{%
1587   \markdownRendererJekyllDataStringPrototype}%
1588 \ExplSyntaxOn
1589 \seq_gput_right:Nn
1590   \g_@@_renderers_seq
1591   { jekyllDataString }
1592 \prop_gput:Nnn
```

```

1593 \g_@@_renderer_arities_prop
1594 { jekyllDataString }
1595 { 2 }
1596 \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.

```

1597 \def\markdownRendererJekyllDataEmpty{%
1598 \markdownRendererJekyllDataEmptyPrototype}%
1599 \ExplSyntaxOn
1600 \seq_gput_right:Nn
1601 \g_@@_renderers_seq
1602 { jekyllDataEmpty }
1603 \prop_gput:Nnn
1604 \g_@@_renderer_arities_prop
1605 { jekyllDataEmpty }
1606 { 1 }
1607 \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.

```

1608 \def\markdownRendererHeadingOne{%
1609 \markdownRendererHeadingOnePrototype}%
1610 \ExplSyntaxOn
1611 \seq_gput_right:Nn
1612 \g_@@_renderers_seq
1613 { headingOne }
1614 \prop_gput:Nnn
1615 \g_@@_renderer_arities_prop
1616 { headingOne }
1617 { 1 }
1618 \ExplSyntaxOff

```

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

```

1619 \def\markdownRendererHeadingTwo{%
1620 \markdownRendererHeadingTwoPrototype}%
1621 \ExplSyntaxOn
1622 \seq_gput_right:Nn
1623 \g_@@_renderers_seq

```



```

1624 { headingTwo }
1625 \prop_gput:Nnn
1626 \g_@@_renderer_arities_prop
1627 { headingTwo }
1628 { 1 }
1629 \ExplSyntaxOff

```

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

```

1630 \def\markdownRendererHeadingThree{%
1631 \markdownRendererHeadingThreePrototype}%
1632 \ExplSyntaxOn
1633 \seq_gput_right:Nn
1634 \g_@@_renderers_seq
1635 { headingThree }
1636 \prop_gput:Nnn
1637 \g_@@_renderer_arities_prop
1638 { headingThree }
1639 { 1 }
1640 \ExplSyntaxOff

```

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

```

1641 \def\markdownRendererHeadingFour{%
1642 \markdownRendererHeadingFourPrototype}%
1643 \ExplSyntaxOn
1644 \seq_gput_right:Nn
1645 \g_@@_renderers_seq
1646 { headingFour }
1647 \prop_gput:Nnn
1648 \g_@@_renderer_arities_prop
1649 { headingFour }
1650 { 1 }
1651 \ExplSyntaxOff

```

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

```

1652 \def\markdownRendererHeadingFive{%
1653 \markdownRendererHeadingFivePrototype}%
1654 \ExplSyntaxOn
1655 \seq_gput_right:Nn
1656 \g_@@_renderers_seq
1657 { headingFive }
1658 \prop_gput:Nnn
1659 \g_@@_renderer_arities_prop
1660 { headingFive }
1661 { 1 }

```

```
1662 \ExplSyntaxOff
```

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

```
1663 \def\markdownRendererHeadingSix{%
1664   \markdownRendererHeadingSixPrototype}%
1665 \ExplSyntaxOn
1666 \seq_gput_right:Nn
1667   \g_@@_renderers_seq
1668   { headingSix }
1669 \prop_gput:Nnn
1670   \g_@@_renderer_arities_prop
1671   { headingSix }
1672   { 1 }
1673 \ExplSyntaxOff
```

2.2.3.20 Thematic Break Renderer The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```
1674 \ExplSyntaxOn
1675 \cs_new:Npn
1676   \markdownRendererThematicBreak
1677   {
1678     \cs_if_exist:NTF
1679       \markdownRendererHorizontalRule
1680       {
1681         \markdownWarning
1682         {
1683           Horizontal~rule~renderer~has~been~deprecated,~
1684           to~be~removed~in~Markdown~3.0.0
1685         }
1686         \markdownRendererHorizontalRule
1687       }
1688     {
1689       \cs_if_exist:NTF
1690         \markdownRendererHorizontalRulePrototype
1691         {
1692           \markdownWarning
1693           {
1694             Horizontal~rule~renderer~prototype~has~been~deprecated,~
1695             to~be~removed~in~Markdown~3.0.0
1696           }
1697           \markdownRendererHorizontalRulePrototype
1698         }
1699     }
```

```

1700         \markdownRendererThematicBreakPrototype
1701     }
1702 }
1703 }
1704 \seq_gput_right:Nn
1705   \g_@@_renderers_seq
1706   { horizontalRule }
1707 \prop_gput:Nnn
1708   \g_@@_renderer_arities_prop
1709   { horizontalRule }
1710   { 0 }
1711 \seq_gput_right:Nn
1712   \g_@@_renderers_seq
1713   { thematicBreak }
1714 \prop_gput:Nnn
1715   \g_@@_renderer_arities_prop
1716   { thematicBreak }
1717   { 0 }
1718 \ExplSyntaxOff

```

2.2.3.21 Note Renderer The `\markdownRendererNote` macro represents a note. This macro will only be produced, when the `notes` option is enabled. The macro receives a single argument that corresponds to the note text.

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

1719 \ExplSyntaxOn
1720 \cs_new:Npn
1721   \markdownRendererNote
1722   {
1723     \cs_if_exist:NTF
1724       \markdownRendererFootnote
1725       {
1726         \markdownWarning
1727         {
1728           Footnote~renderer~has~been~deprecated,~
1729           to~be~removed~in~Markdown~3.0.0
1730         }
1731       }
1732     }
1733   {
1734     \cs_if_exist:NTF
1735       \markdownRendererFootnotePrototype
1736       {
1737         \markdownWarning
1738         {
1739           Footnote~renderer~prototype~has~been~deprecated,~

```

```

1740         to~be~removed~in~Markdown~3.0.0
1741     }
1742     \markdownRendererFootnotePrototype
1743 }
1744 {
1745     \markdownRendererNotePrototype
1746 }
1747 }
1748 }
1749 \seq_gput_right:Nn
1750 \g_@@_renderers_seq
1751 { footnote }
1752 \prop_gput:Nnn
1753 \g_@@_renderer_arities_prop
1754 { footnote }
1755 { 1 }
1756 \seq_gput_right:Nn
1757 \g_@@_renderers_seq
1758 { note }
1759 \prop_gput:Nnn
1760 \g_@@_renderer_arities_prop
1761 { note }
1762 { 1 }
1763 \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.

```

1764 \def\markdownRendererCite{%
1765     \markdownRendererCitePrototype}%
1766 \ExplSyntaxOn
1767 \seq_gput_right:Nn
1768 \g_@@_renderers_seq
1769 { cite }
1770 \prop_gput:Nnn
1771 \g_@@_renderer_arities_prop
1772 { cite }
1773 { 1 }
1774 \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.

```

1775 \def\markdownRendererTextCite{%
1776   \markdownRendererTextCitePrototype}%
1777 \ExplSyntaxOn
1778 \seq_gput_right:Nn
1779   \g_@@_renderers_seq
1780   { textCite }
1781 \prop_gput:Nnn
1782   \g_@@_renderer_arities_prop
1783   { textCite }
1784   { 1 }
1785 \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.

```

1786 \def\markdownRendererTable{%
1787   \markdownRendererTablePrototype}%
1788 \ExplSyntaxOn
1789 \seq_gput_right:Nn
1790   \g_@@_renderers_seq
1791   { table }
1792 \prop_gput:Nnn
1793   \g_@@_renderer_arities_prop
1794   { table }
1795   { 3 }
1796 \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.

```

1797 \def\markdownRendererInlineHtmlComment{%
1798   \markdownRendererInlineHtmlCommentPrototype}%
1799 \ExplSyntaxOn
1800 \seq_gput_right:Nn
1801   \g_@@_renderers_seq
1802   { inlineHtmlComment }
1803 \prop_gput:Nnn
1804   \g_@@_renderer_arities_prop
1805   { inlineHtmlComment }
1806   { 1 }
1807 \ExplSyntaxOff
1808 \def\markdownRendererBlockHtmlCommentBegin{%
1809   \markdownRendererBlockHtmlCommentBeginPrototype}%
1810 \ExplSyntaxOn
1811 \seq_gput_right:Nn
1812   \g_@@_renderers_seq
1813   { blockHtmlCommentBegin }
1814 \prop_gput:Nnn
1815   \g_@@_renderer_arities_prop
1816   { blockHtmlCommentBegin }
1817   { 0 }
1818 \ExplSyntaxOff
1819 \def\markdownRendererBlockHtmlCommentEnd{%
1820   \markdownRendererBlockHtmlCommentEndPrototype}%
1821 \ExplSyntaxOn
1822 \seq_gput_right:Nn
1823   \g_@@_renderers_seq
1824   { blockHtmlCommentEnd }
1825 \prop_gput:Nnn
1826   \g_@@_renderer_arities_prop
1827   { blockHtmlCommentEnd }
1828   { 0 }
1829 \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.

```

1830 \def\markdownRendererInlineHtmlTag{%
1831   \markdownRendererInlineHtmlTagPrototype}%
1832 \ExplSyntaxOn
1833 \seq_gput_right:Nn
1834   \g_@@_renderers_seq
1835   { inlineHtmlTag }
1836 \prop_gput:Nnn
1837   \g_@@_renderer_arities_prop
1838   { inlineHtmlTag }
1839   { 1 }
1840 \ExplSyntaxOff
1841 \def\markdownRendererInputBlockHtmlElement{%
1842   \markdownRendererInputBlockHtmlElementPrototype}%
1843 \ExplSyntaxOn
1844 \seq_gput_right:Nn
1845   \g_@@_renderers_seq
1846   { inputBlockHtmlElement }
1847 \prop_gput:Nnn
1848   \g_@@_renderer_arities_prop
1849   { inputBlockHtmlElement }
1850   { 1 }
1851 \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="⟨identifier⟩"` in HTML and `#⟨identifier⟩` in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the $\langle identifier \rangle$.

`\markdownRendererAttributeName` represents the $\langle class name \rangle$ of a markdown element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` 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.

```

1852 \def\markdownRendererAttributeIdentifier{%
1853   \markdownRendererAttributeIdentifierPrototype}%
1854 \ExplSyntaxOn
1855 \seq_gput_right:Nn
1856   \g_@@_renderers_seq
1857   { attributeIdentifier }
1858 \prop_gput:Nnn
1859   \g_@@_renderer_arities_prop
1860   { attributeIdentifier }

```

```

1861 { 1 }
1862 \ExplSyntaxOff
1863 \def\markdownRendererAttributeClassName{%
1864 \markdownRendererAttributeClassNamePrototype}%
1865 \ExplSyntaxOn
1866 \seq_gput_right:Nn
1867 \g_@@_renderers_seq
1868 { attributeClassName }
1869 \prop_gput:Nnn
1870 \g_@@_renderer_arities_prop
1871 { attributeClassName }
1872 { 1 }
1873 \ExplSyntaxOff
1874 \def\markdownRendererAttributeKeyValue{%
1875 \markdownRendererAttributeKeyValuePrototype}%
1876 \ExplSyntaxOn
1877 \seq_gput_right:Nn
1878 \g_@@_renderers_seq
1879 { attributeKeyValue }
1880 \prop_gput:Nnn
1881 \g_@@_renderer_arities_prop
1882 { attributeKeyValue }
1883 { 2 }
1884 \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.

```

1885 \def\markdownRendererHeaderAttributeContextBegin{%
1886 \markdownRendererHeaderAttributeContextBeginPrototype}%
1887 \ExplSyntaxOn
1888 \seq_gput_right:Nn
1889 \g_@@_renderers_seq
1890 { headerAttributeContextBegin }
1891 \prop_gput:Nnn
1892 \g_@@_renderer_arities_prop
1893 { headerAttributeContextBegin }
1894 { 0 }
1895 \ExplSyntaxOff
1896 \def\markdownRendererHeaderAttributeContextEnd{%
1897 \markdownRendererHeaderAttributeContextEndPrototype}%
1898 \ExplSyntaxOn
1899 \seq_gput_right:Nn
1900 \g_@@_renderers_seq

```



```

1901 { headerAttributeContextEnd }
1902 \prop_gput:Nnn
1903 \g_@@_renderer_arities_prop
1904 { headerAttributeContextEnd }
1905 { 0 }
1906 \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.

```

1907 \def\markdownRendererStrikeThrough{%
1908 \markdownRendererStrikeThroughPrototype}%
1909 \ExplSyntaxOn
1910 \seq_gput_right:Nn
1911 \g_@@_renderers_seq
1912 { strikeThrough }
1913 \prop_gput:Nnn
1914 \g_@@_renderer_arities_prop
1915 { strikeThrough }
1916 { 1 }
1917 \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.

```

1918 \def\markdownRendererSuperscript{%
1919 \markdownRendererSuperscriptPrototype}%
1920 \ExplSyntaxOn
1921 \seq_gput_right:Nn
1922 \g_@@_renderers_seq
1923 { superscript }
1924 \prop_gput:Nnn
1925 \g_@@_renderer_arities_prop
1926 { superscript }
1927 { 1 }
1928 \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.

```

1929 \def\markdownRendererSubscript{%
1930   \markdownRendererSubscriptPrototype}%
1931 \ExplSyntaxOn
1932 \seq_gput_right:Nn
1933   \g_@@_renderers_seq
1934   { subscript }
1935 \prop_gput:Nnn
1936   \g_@@_renderer_arities_prop
1937   { subscript }
1938   { 1 }

```

2.2.3.32 Raw Content Renderers The `\markdownRendererInputRawInline` macro represents an inline raw span. The macro receives two arguments: the filename of a file containing the inline raw span contents and the raw attribute that designates the format of the inline raw span. This macro will only be produced, when the `rawAttribute` option is enabled.

```

1939 \def\markdownRendererInputRawInline{%
1940   \markdownRendererInputRawInlinePrototype}%
1941 \ExplSyntaxOn
1942 \seq_gput_right:Nn
1943   \g_@@_renderers_seq
1944   { inputRawInline }
1945 \prop_gput:Nnn
1946   \g_@@_renderer_arities_prop
1947   { inputRawInline }
1948   { 2 }
1949 \ExplSyntaxOff

```

The `\markdownRendererInputRawBlock` macro represents a raw block. The macro receives two arguments: the filename of a file containing the raw block and the raw attribute that designates the format of the raw block. This macro will only be produced, when the `rawAttribute` and `fencedCode` options are enabled.

```

1950 \def\markdownRendererInputRawBlock{%
1951   \markdownRendererInputRawBlockPrototype}%
1952 \ExplSyntaxOn
1953 \seq_gput_right:Nn
1954   \g_@@_renderers_seq
1955   { inputRawBlock }
1956 \prop_gput:Nnn
1957   \g_@@_renderer_arities_prop
1958   { inputRawBlock }
1959   { 2 }
1960 \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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$ kernel.

```
1961 \ExplSyntaxOn
1962 \keys_define:nn
1963   { markdown/jekyllData }
1964   { }
1965 \ExplSyntaxOff
```

The following $\text{T}_{\text{E}}\text{X}$ 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ and $\text{C}^{\text{O}}\text{N}\text{T}_{\text{E}}\text{X}\text{T}$ implementations (see sections 3.3 and 3.4).

```
1966 \ExplSyntaxOn
1967 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
1968   {
1969     \seq_map_function:NN
1970       \g_@@_renderers_seq
1971       \@@_plaintex_define_renderer_prototype:n
1972     \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
1973     \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1974     \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1975     \let\markdownRendererBlockHtmlCommentEnd=\fi
```

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```
1976   \cs_undefine:N \markdownRendererFootnote
1977   \cs_undefine:N \markdownRendererFootnotePrototype
```

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```
1978   \cs_undefine:N \markdownRendererHorizontalRule
1979   \cs_undefine:N \markdownRendererHorizontalRulePrototype
1980 }
1981 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
1982   {
1983     \@@_renderer_prototype_tl_to_csname:nN
1984     { #1 }
1985     \l_tmpa_tl
1986     \prop_get:NnN
1987       \g_@@_renderer_arities_prop
1988       { #1 }
1989     \l_tmpb_tl
1990     \@@_plaintex_define_renderer_prototype:cV
```

```

1991     { \l_tmpa_tl }
1992     \l_tmpb_tl
1993   }
1994 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
1995 {
1996   \tl_set:Nn
1997     \l_tmpa_tl
1998     { \str_uppercase:n { #1 } }
1999   \tl_set:Nx
2000     #2
2001     {
2002       markdownRenderer
2003       \tl_head:f { \l_tmpa_tl }
2004       \tl_tail:n { #1 }
2005       Prototype
2006     }
2007 }
2008 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
2009 {
2010   \cs_generate_from_arg_count:NNnn
2011     #1
2012     \cs_set:Npn
2013     { #2 }
2014     { }
2015 }
2016 \cs_generate_variant:Nn
2017   \@@_plaintex_define_renderer_prototype:Nn
2018   { cV }
2019 \@@_plaintex_define_renderer_prototypes:
2020 \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 \TeX engine that does not support direct Lua access is starting to buffer a text. The plain \TeX implementation changes the category code of plain \TeX special characters to other,

but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
2021 \let\markdownMakeOther\relax
```

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

```
2022 \let\markdownReadAndConvert\relax
```

```
2023 \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*.

```
2024 \catcode`\|=0\catcode`\=12%
```

```
2025 |gdef|markdownBegin{%
```

```
2026   |markdownReadAndConvert{\markdownEnd}%
```

```
2027   {|\markdownEnd}}%
```

```
2028 |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.

```
2029 \ExplSyntaxOn
```

```
2030 \cs_if_exist:NF
```

```
2031   \markdownMode
```

```
2032   {
```

```
2033     \file_if_exist:nTF
```

```
2034       { lt3luabridge.tex }
```

```

2035     {
2036       \cs_new:Npn
2037         \markdownMode
2038         { 3 }
2039     }
2040     {
2041       \cs_if_exist:NTF
2042         \directlua
2043         {
2044           \cs_new:Npn
2045             \markdownMode
2046             { 2 }
2047         }
2048         {
2049           \cs_new:Npn
2050             \markdownMode
2051             { 0 }
2052         }
2053     }
2054 }

```

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

```

2055 \def\markdownLuaRegisterIBCallback#1{\relax}%
2056 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

2.3 L^AT_EX Interface

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

The L^AT_EX implementation redefines the plain T_EX logging macros (see Section 3.2.1) to use the L^AT_EX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

2057 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2058 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2059 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
2060 \input markdown/markdown

```

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

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

where $\langle options \rangle$ are the \LaTeX interface options (see Section 2.3.2). Note that $\langle options \rangle$ 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 $\LaTeX 2_\epsilon$ parses package options.

2.3.1 Typesetting Markdown

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

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

```
2061 \newenvironment{markdown}\relax\relax
2062 \newenvironment{markdown*}[1]\relax\relax
```

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

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

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

<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 \TeX . Unlike the `\markdownInput` macro provided by the plain \TeX interface, this macro also accepts \LaTeX interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

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

```

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

```

2.3.2 Options

The \LaTeX 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 \LaTeX themes described in Section 2.3.2.2, and the \LaTeX setup snippets described in Section 2.3.2.3, \LaTeX options map directly to the options recognized by the plain \TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain \TeX interface (see Sections 2.2.3 and 2.2.4).

The \LaTeX options may be specified when loading the \LaTeX package, when using the `markdown*` \LaTeX 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:

```

2063 \ExplSyntaxOn
2064 \cs_new:Nn
2065   \@@_setup:n
2066   {
2067     \keys_set:nn
2068       { markdown/latex-options }
2069       { #1 }
2070   }
2071 \let\markdownSetup=\@@_setup:n
2072 \ExplSyntaxOff

```

We may also store \LaTeX 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:

```

2073 \newcommand\markdownSetupSnippet[2]{%
2074   \markdownIfSnippetExists{#1}%
2075   {%
2076     \markdownWarning
2077     {Redefined setup snippet \markdownLaTeXThemeName#1}%
2078     \csname markdownLaTeXSetupSnippet%
2079       \markdownLaTeXThemeName#1\endcsname={#2}%
2080   }{%
2081     \newtoks\next

```



```

2082     \next={#2}%
2083     \expandafter\let\csgname markdownLaTeXSetupSnippet%
2084     \markdownLaTeXThemeName#1\endcsname=\next
2085   }}%

```

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

```

2086 \newcommand\markdownIfSnippetExists[3]{%
2087   \@ifundefined
2088     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
2089     {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and L^AT_EX themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of L^AT_EX options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

2090 \ExplSyntaxOn
2091 \seq_new:N \g_@@_latex_options_seq

```

To enable the reflection of default L^AT_EX options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```

2092 \prop_new:N \g_@@_latex_option_types_prop
2093 \prop_new:N \g_@@_default_latex_options_prop
2094 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
2095 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
2096 \cs_new:Nn
2097   \@@_add_latex_option:nnn
2098   {
2099     \@@_add_option:Vnnn
2100     \c_@@_option_layer_latex_tl
2101     { #1 }
2102     { #2 }
2103     { #3 }
2104   }

```

2.3.2.1 No default token renderer prototypes Default token renderer prototypes require L^AT_EX 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_EX implementation (see Section 3.2.2) and prevent the soft L^AT_EX 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}
```

```

2105 \@_add_latex_option:nnn
2106   { plain }
2107   { boolean }
2108   { false }
2109 \ExplSyntaxOff

```

2.3.2.2 L^AT_EX themes User-defined L^AT_EX themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L^AT_EX packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The L^AT_EX option `theme=<theme name>` loads a L^AT_EX 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 L^AT_EX package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single L^AT_EX document class or for a single L^AT_EX package. The preferred format of a theme name is `<theme author>/<target 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 L^AT_EX packages are identified only by their filenames, not by their pathnames. [9] 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 L^AT_EX package named `markdownthemewitiko_beamer_MU.sty`.

If the L^AT_EX 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 L^AT_EX 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` L^AT_EX package, and finally the `markdownthemewitiko_dot.sty` L^AT_EX package:

```

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

```

```

2110 \newif\ifmarkdownLaTeXLoaded
2111 \markdownLaTeXLoadedfalse
2112 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
2113 \ExplSyntaxOn
2114 \tl_new:N \markdownLaTeXThemePackageName
2115 \cs_new:Nn
2116 \@@_set_latex_theme:n
2117 {
2118   \str_if_in:nnF
2119     { #1 }
2120     { / }
2121     {
2122       \markdownError
2123       { Won't~load~theme~with~unqualified~name~#1 }
2124       { Theme~names~must~contain~at~least~one~forward~slash }
2125     }
2126   \str_if_in:nnT
2127     { #1 }
2128     { _ }
2129     {
2130       \markdownError
2131       { Won't~load~theme~with~an~underscore~in~its~name~#1 }
2132       { Theme~names~must~not~contain~underscores~in~their~names }
2133     }
2134   \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
2135   \str_replace_all:Nnn
2136     \markdownLaTeXThemePackageName
2137     { / }
2138     { _ }
2139   \edef\markdownLaTeXThemePackageName{
2140     markdowntheme\markdownLaTeXThemePackageName}
2141   \expandafter\markdownLaTeXThemeLoad\expandafter{
2142     \markdownLaTeXThemePackageName}{#1/}
2143 }
2144 \keys_define:nn
2145 { markdown/latex-options }
2146 {
2147   theme .code:n = { \@@_set_latex_theme:n { #1 } },
2148 }
2149 \ExplSyntaxOff

```

The L^AT_EX 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 $\langle theme\ name \rangle/$ 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^AT_EX, themes may not be loaded after the beginning of a L^AT_EX document.

```
2150 \ExplSyntaxOn
2151 \@onlypreamble
2152   \@@_set_latex_theme:n
2153 \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;

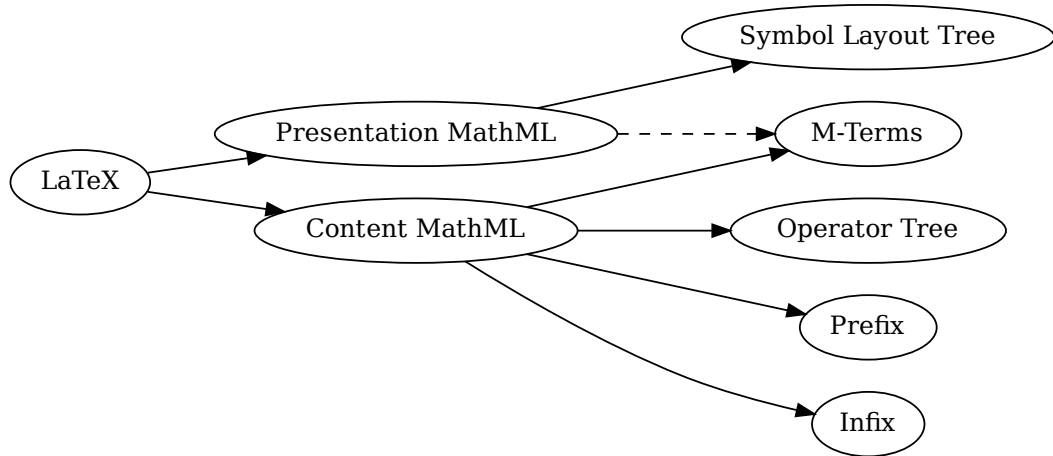
 latex [label = "LaTeX"];
 pmml [label = "Presentation MathML"];
 cmml [label = "Content MathML"];
 slt [label = "Symbol Layout Tree"];
 opt [label = "Operator Tree"];
 prefix [label = "Prefix"];
 infix [label = "Infix"];
 mterms [label = "M-Terms"];
```

```

}
...
\end{markdown}
\end{document}

```

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



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

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

```

2154 \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}

\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 5. The

```

\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**

theme requires the catchfile L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X option is enabled.

2155 \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{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”.

2156 \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  $\langle value \rangle$ :

```
2157 \ExplSyntaxOn
2158 \keys_define:nn
2159 { markdown/latex-options }
2160 {
2161 snippet .code:n = {
2162 \markdownIfSnippetExists{#1}
2163 {
2164 \expandafter\markdownSetup\expandafter{
2165 \the\csname markdownLaTeXSetupSnippet
2166 \markdownLaTeXThemeName#1\endcsname}
2167 }{
2168 \markdownError
2169 {Can't~invoke~setup~snippet~#1}
2170 {The~setup~snippet~is~undefined}
2171 }
2172 }
2173 }
2174 \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.

```
2175 \ExplSyntaxOn
2176 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
2177 {
2178 \seq_map_inline:Nn
2179 \g_@@_latex_options_seq
2180 {
2181 \@@_plain_tex_define_option_command:n
2182 { ##1 }
2183 }

```

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.

```
2184 \seq_map_inline:Nn
2185 \g_@@_option_layers_seq
2186 {
2187 \seq_map_inline:cn
2188 { g_@@_ ##1 _options_seq }
2189 }

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [5] also show that `snake_case` is faster to read than camelCase.

```
2190 \@@_with_various_cases:nn
2191 { #####1 }
2192 {
2193 \@@_latex_define_option_keyval:nnn
2194 { ##1 }
2195 { #####1 }
2196 { #####1 }
2197 }
2198 }
2199 }
2200 }
2201 \cs_new:Nn \@@_latex_define_option_keyval:nnn
2202 {
2203 \prop_get:cnN
2204 { g_@@_ #1 _option_types_prop }
2205 { #2 }
2206 \l_tmpa_tl
2207 \keys_define:nn

```



```

2208 { markdown/latex-options }
2209 {
2210 #3 .code:n = {
2211 \@@_set_option_value:nn
2212 { #2 }
2213 { ##1 }
2214 },
2215 }
2216 \str_if_eq:VVT
2217 \l_tmpa_tl
2218 \c_@@_option_type_boolean_tl
2219 {
2220 \keys_define:nn
2221 { markdown/latex-options }
2222 {
2223 #3 .default:n = { true },
2224 }
2225 }

```

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.

```

2226 \str_if_eq:VVT
2227 \l_tmpa_tl
2228 \c_@@_option_type_clist_tl
2229 {
2230 \tl_set:Nn
2231 \l_tmpa_tl
2232 { #3 }
2233 \tl_reverse:N
2234 \l_tmpa_tl
2235 \str_if_eq:enF
2236 {
2237 \tl_head:V
2238 \l_tmpa_tl
2239 }
2240 { s }
2241 {
2242 \msg_error:nnn
2243 { @@ }
2244 { malformed-name-for-clist-option }
2245 { #3 }
2246 }
2247 \tl_set:Nx
2248 \l_tmpa_tl

```

```

2249 {
2250 \tl_tail:V
2251 \l_tmpa_tl
2252 }
2253 \tl_reverse:N
2254 \l_tmpa_tl
2255 \tl_put_right:Nn
2256 \l_tmpa_tl
2257 {
2258 .code:n = {
2259 \@@_get_option_value:nN
2260 { #2 }
2261 \l_tmpa_tl
2262 \clist_set:NV
2263 \l_tmpa_clist
2264 { \l_tmpa_tl, { ##1 } }
2265 \@@_set_option_value:nV
2266 { #2 }
2267 \l_tmpa_clist
2268 }
2269 }
2270 \keys_define:nV
2271 { markdown/latex-options }
2272 \l_tmpa_tl
2273 }
2274 }
2275 \cs_generate_variant:Nn
2276 \clist_set:Nn
2277 { NV }
2278 \cs_generate_variant:Nn
2279 \keys_define:nn
2280 { nV }
2281 \cs_generate_variant:Nn
2282 \@@_set_option_value:nn
2283 { nV }
2284 \prg_generate_conditional_variant:Nnn
2285 \str_if_eq:nn
2286 { en }
2287 { F }
2288 \msg_new:nnn
2289 { @@ }
2290 { malformed-name-for-clist-option }
2291 {
2292 Clist~option~name~#1~does~not~end~with~-s.
2293 }
2294 \@@_latex_define_option_commands_and_keyvals:
2295 \ExplSyntaxOff

```

The `\markdownOptionFinalizeCache` and `\markdownOptionFrozenCache` plain  $\TeX$  options are exposed through  $\LaTeX$  options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [10, 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  $\LaTeX$  document sources for distribution.

```
2296 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2297 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

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

```
2298 \ExplSyntaxOn
2299 \cs_new:Nn \@@_latex_define_renderers:
2300 {
2301 \seq_map_function:NN
2302 \g_@@_renderers_seq
2303 \@@_latex_define_renderer:n
2304 }
2305 \cs_new:Nn \@@_latex_define_renderer:n
```

```

2306 {
2307 \@@_renderer_tl_to_csname:nN
2308 { #1 }
2309 \l_tmpa_tl
2310 \prop_get:NnN
2311 \g_@@_renderer_arities_prop
2312 { #1 }
2313 \l_tmpb_tl
2314 \@@_latex_define_renderer:ncV
2315 { #1 }
2316 { \l_tmpa_tl }
2317 \l_tmpb_tl
2318 }
2319 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2320 {
2321 \tl_set:Nn
2322 \l_tmpa_tl
2323 { \str_uppercase:n { #1 } }
2324 \tl_set:Nx
2325 #2
2326 {
2327 markdownRenderer
2328 \tl_head:f { \l_tmpa_tl }
2329 \tl_tail:n { #1 }
2330 }
2331 }
2332 \cs_new:Nn \@@_latex_define_renderer:nNn
2333 {
2334 \@@_with_various_cases:nn
2335 { #1 }
2336 {
2337 \keys_define:nn
2338 { markdown/latex-options/renderers }
2339 {
2340 ##1 .code:n = {
2341 \cs_generate_from_arg_count:NNnn
2342 #2
2343 \cs_set:Npn
2344 { #3 }
2345 { #####1 }
2346 },
2347 }
2348 }
2349 }
2350 \cs_generate_variant:Nn
2351 \@@_latex_define_renderer:nNn
2352 { ncV }

```

2353 \ExplSyntaxOff

The following example L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X Markdown Token Renderer Prototypes** The L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X interface (see Section 2.2.4).

```
2354 \ExplSyntaxOn
2355 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2356 {
2357 \seq_map_function:NN
2358 \g_@@_renderers_seq
2359 \@@_latex_define_renderer_prototype:n
2360 }
2361 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2362 {
2363 \@@_renderer_prototype_tl_to_csname:nN
2364 { #1 }
2365 \l_tmpa_tl
2366 \prop_get:NnN
2367 \g_@@_renderer_arities_prop
2368 { #1 }
2369 \l_tmpb_tl
2370 \@@_latex_define_renderer_prototype:ncV
2371 { #1 }
2372 { \l_tmpa_tl }
2373 \l_tmpb_tl
2374 }
2375 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2376 {
2377 \@@_with_various_cases:nn
2378 { #1 }
2379 {
2380 \keys_define:nn
2381 { markdown/latex-options/renderer-prototypes }

```

```

2382 {
2383 ##1 .code:n = {
2384 \cs_generate_from_arg_count:NNnn
2385 #2
2386 \cs_set:Npn
2387 { #3 }
2388 { ####1 }
2389 },
2390 }
2391 }
2392 }
2393 \cs_generate_variant:Nn
2394 @@_latex_define_renderer_prototype:nNn
2395 { ncV }
2396 \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).

```

2397 \writestatus{loading}{ConTeXt User Module / markdown}%
2398 \startmodule[markdown]
2399 \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.

```

2400 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2401 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2402 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2403 \do\#\do\^\do_do\%do\~}%
2404 \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  $\TeX$  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.

```
2405 \let\startmarkdown\relax
```

```
2406 \let\stopmarkdown\relax
```

```
2407 \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  $\TeX$  interface.

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

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

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

The following example L $\TeX$  code showcases the usage of the  $\markdownInput$  macro:

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

## 2.4.2 Options

The ConT<sub>E</sub>Xt 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.

ConT<sub>E</sub>Xt options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2).

The ConT<sub>E</sub>Xt 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:

```
2408 \ExplSyntaxOn
2409 \cs_new:Nn
2410 \@@_setup:n
2411 {
2412 \keys_set:nn
2413 { markdown/context-options }
2414 { #1 }
2415 }
2416 \long\def\setupmarkdown[#1]
2417 {
2418 \@@_setup:n
2419 { #1 }
2420 }
2421 \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.

```
2422 \ExplSyntaxOn
2423 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2424 {
2425 \seq_map_inline:Nn
2426 \g_@@_option_layers_seq
2427 {
2428 \seq_map_inline:cn
2429 { g_@@_ ##1 _options_seq }
2430 {
```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```
2431 \@@_with_various_cases:nn
2432 { ###1 }
2433 {
2434 \@@_context_define_option_keyval:nnn
```



```

2435 { ##1 }
2436 { #####1 }
2437 { #####1 }
2438 }
2439 }
2440 }
2441 }

```

Furthermore, we also accept caseless variants of options in line with the style of ConTeXt.

```

2442 \cs_new:Nn \@@_caseless:N
2443 {
2444 \regex_replace_all:nnN
2445 { ([a-z])([A-Z]) }
2446 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
2447 #1
2448 \tl_set:Nx
2449 #1
2450 { #1 }
2451 }
2452 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
2453 \cs_new:Nn \@@_context_define_option_keyval:nnn
2454 {
2455 \prop_get:cnN
2456 { g_@@_ #1 _option_types_prop }
2457 { #2 }
2458 \l_tmpa_tl
2459 \keys_define:nn
2460 { markdown/context-options }
2461 {
2462 #3 .code:n = {
2463 \tl_set:Nx
2464 \l_tmpa_tl
2465 {
2466 \str_case:nnF
2467 { ##1 }
2468 {
2469 { yes } { true }
2470 { no } { false }
2471 }
2472 { ##1 }
2473 }
2474 \@@_set_option_value:nV
2475 { #2 }
2476 \l_tmpa_tl
2477 },
2478 }

```

```

2479 \str_if_eq:VVT
2480 \l_tmpa_tl
2481 \c_@@_option_type_boolean_tl
2482 {
2483 \keys_define:nn
2484 { markdown/context-options }
2485 {
2486 #3 .default:n = { true },
2487 }
2488 }
2489 }
2490 \cs_generate_variant:Nn
2491 \@@_set_option_value:nn
2492 { nV }
2493 \@@_context_define_option_commands_and_keyvals:
2494 \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).

```

2495 local upper, gsub, format, length =
2496 string.upper, string.gsub, string.format, string.len
2497 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2498 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2499 lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)

```

### 3.1.1 Utility Functions

This section documents the utility functions used by the plain T<sub>E</sub>X 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.

```
2500 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.

```
2501 function util.err(msg, exit_code)
2502 io.stderr:write("markdown.lua: " .. msg .. "\n")
2503 os.exit(exit_code or 1)
2504 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.

```
2505 function util.cache(dir, string, salt, transform, suffix)
2506 local digest = md5.sumhexa(string .. (salt or ""))
2507 local name = util.pathname(dir, digest .. suffix)
2508 local file = io.open(name, "r")
2509 if file == nil then -- If no cache entry exists, then create a new one.
2510 file = assert(io.open(name, "w"),
2511 [[Could not open file]] .. name .. [[for writing]])
2512 local result = string
2513 if transform ~= nil then
2514 result = transform(result)
2515 end
2516 assert(file:write(result))
2517 assert(file:close())
2518 end
2519 return name
2520 end
```

The `util.cache_verbatim` method strips whitespaces from the end of `string` and calls `util.cache` with `dir`, `string`, no salt or transformations, and the `.verbatim` suffix.

```
2521 function util.cache_verbatim(dir, string)
2522 string = string:gsub('[\r\n%s]*$', '')
2523 local name = util.cache(dir, string, nil, nil, ".verbatim")
2524 return name
2525 end
```

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

```
2526 function util.table_copy(t)
2527 local u = { }
```

```

2528 for k, v in pairs(t) do u[k] = v end
2529 return setmetatable(u, getmetatable(t))
2530 end

```

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

```

2531 function util.encode_json_string(s)
2532 s = s:gsub([[\\]], [[\\]])
2533 s = s:gsub([[\"]], [[\"]])
2534 return [[\"]] .. s .. [[\"]]
2535 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]

```

2536 util.lookup_files = (function()
2537 local ran_ok, kpse = pcall(require, "kpse")
2538 if ran_ok then
2539 kpse.set_program_name("luatex")
2540 else
2541 kpse = { lookup = function(f, _) return f end }
2542 end
2543
2544 local function lookup_files(f, options)
2545 return kpse.lookup(f, options)
2546 end
2547
2548 return lookup_files
2549 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 Ierusalimsky [11, Chapter 21].

```

2550 function util.expand_tabs_in_line(s, tabstop)
2551 local tab = tabstop or 4
2552 local corr = 0
2553 return (s:gsub("(\\t)", function(p)
2554 local sp = tab - (p - 1 + corr) % tab
2555 corr = corr - 1 + sp
2556 return string.rep(" ", sp)
2557 end))
2558 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.

```

2559 function util.walk(t, f)
2560 local typ = type(t)
2561 if typ == "string" then
2562 f(t)
2563 elseif typ == "table" then
2564 local i = 1
2565 local n
2566 n = t[i]
2567 while n do
2568 util.walk(n, f)
2569 i = i + 1
2570 n = t[i]
2571 end
2572 elseif typ == "function" then
2573 local ok, val = pcall(t)
2574 if ok then
2575 util.walk(val, f)
2576 end
2577 else
2578 f(tostring(t))
2579 end
2580 end

```

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

```

2581 function util.flatten(ary)
2582 local new = {}
2583 for _,v in ipairs(ary) do
2584 if type(v) == "table" then
2585 for _,w in ipairs(util.flatten(v)) do
2586 new[#new + 1] = w
2587 end
2588 else
2589 new[#new + 1] = v
2590 end
2591 end
2592 return new
2593 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.

```

2594 function util.rope_to_string(rope)
2595 local buffer = {}
2596 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2597 return table.concat(buffer)
2598 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.

```
2599 function util.rope_last(rope)
2600 if #rope == 0 then
2601 return nil
2602 else
2603 local l = rope[#rope]
2604 if type(l) == "table" then
2605 return util.rope_last(l)
2606 else
2607 return l
2608 end
2609 end
2610 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  $1 \leq i \leq \#ary$ .

```
2611 function util.intersperse(ary, x)
2612 local new = {}
2613 local l = #ary
2614 for i,v in ipairs(ary) do
2615 local n = #new
2616 new[n + 1] = v
2617 if i ~= l then
2618 new[n + 2] = x
2619 end
2620 end
2621 return new
2622 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  $1 \leq i \leq \#ary$ .

```
2623 function util.map(ary, f)
2624 local new = {}
2625 for i,v in ipairs(ary) do
2626 new[i] = f(v)
2627 end
2628 return new
2629 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.

```
2630 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
2631 local char_escapes_list = ""
2632 for i,_ in pairs(char_escapes) do
2633 char_escapes_list = char_escapes_list .. i
2634 end
```

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

```
2635 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.

```
2636 if string_escapes then
2637 for k,v in pairs(string_escapes) do
2638 escapable = P(k) / v + escapable
2639 end
2640 end
```

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

```
2641 local escape_string = Cs((escapable + any)^0)
```

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

```
2642 return function(s)
2643 return lpeg.match(escape_string, s)
2644 end
2645 end
```

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

```
2646 function util.pathname(dir, file)
2647 if #dir == 0 then
2648 return file
2649 else
2650 return dir .. "/" .. file
2651 end
2652 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.

```
2653 local entities = {}
2654
2655 local character_entities = {
2656 ["Tab"] = 9,
2657 ["NewLine"] = 10,
2658 ["excl"] = 33,
2659 ["quot"] = 34,
2660 ["QUOT"] = 34,
2661 ["num"] = 35,
2662 ["dollar"] = 36,
2663 ["percent"] = 37,
2664 ["amp"] = 38,
2665 ["AMP"] = 38,
2666 ["apos"] = 39,
2667 ["lpar"] = 40,
2668 ["rpar"] = 41,
2669 ["ast"] = 42,
2670 ["midast"] = 42,
2671 ["plus"] = 43,
2672 ["comma"] = 44,
2673 ["period"] = 46,
2674 ["sol"] = 47,
2675 ["colon"] = 58,
2676 ["semi"] = 59,
2677 ["lt"] = 60,
2678 ["LT"] = 60,
2679 ["equals"] = 61,
2680 ["gt"] = 62,
2681 ["GT"] = 62,
2682 ["quest"] = 63,
2683 ["commat"] = 64,
2684 ["lsqb"] = 91,
2685 ["lbrack"] = 91,
2686 ["bsol"] = 92,
2687 ["rsqb"] = 93,
2688 ["rbrack"] = 93,
2689 ["Hat"] = 94,
2690 ["lowbar"] = 95,
2691 ["grave"] = 96,
2692 ["DiacriticalGrave"] = 96,
2693 ["lcub"] = 123,
2694 ["lbrace"] = 123,
```



2695 ["verbar"] = 124,  
2696 ["vert"] = 124,  
2697 ["VerticalLine"] = 124,  
2698 ["rcub"] = 125,  
2699 ["rbrace"] = 125,  
2700 ["nbsp"] = 160,  
2701 ["NonBreakingSpace"] = 160,  
2702 ["iexcl"] = 161,  
2703 ["cent"] = 162,  
2704 ["pound"] = 163,  
2705 ["curren"] = 164,  
2706 ["yen"] = 165,  
2707 ["brvbar"] = 166,  
2708 ["sect"] = 167,  
2709 ["Dot"] = 168,  
2710 ["die"] = 168,  
2711 ["DoubleDot"] = 168,  
2712 ["uml"] = 168,  
2713 ["copy"] = 169,  
2714 ["COPY"] = 169,  
2715 ["ordf"] = 170,  
2716 ["laquo"] = 171,  
2717 ["not"] = 172,  
2718 ["shy"] = 173,  
2719 ["reg"] = 174,  
2720 ["circledR"] = 174,  
2721 ["REG"] = 174,  
2722 ["macr"] = 175,  
2723 ["OverBar"] = 175,  
2724 ["strns"] = 175,  
2725 ["deg"] = 176,  
2726 ["plusmn"] = 177,  
2727 ["pm"] = 177,  
2728 ["PlusMinus"] = 177,  
2729 ["sup2"] = 178,  
2730 ["sup3"] = 179,  
2731 ["acute"] = 180,  
2732 ["DiacriticalAcute"] = 180,  
2733 ["micro"] = 181,  
2734 ["para"] = 182,  
2735 ["middot"] = 183,  
2736 ["centerdot"] = 183,  
2737 ["CenterDot"] = 183,  
2738 ["cedil"] = 184,  
2739 ["Cedilla"] = 184,  
2740 ["sup1"] = 185,  
2741 ["ordm"] = 186,

2742 ["raquo"] = 187,  
2743 ["frac14"] = 188,  
2744 ["frac12"] = 189,  
2745 ["half"] = 189,  
2746 ["frac34"] = 190,  
2747 ["iquest"] = 191,  
2748 ["Agrave"] = 192,  
2749 ["Aacute"] = 193,  
2750 ["Acirc"] = 194,  
2751 ["Atilde"] = 195,  
2752 ["Auml"] = 196,  
2753 ["Aring"] = 197,  
2754 ["AElig"] = 198,  
2755 ["Ccedil"] = 199,  
2756 ["Egrave"] = 200,  
2757 ["Eacute"] = 201,  
2758 ["Ecirc"] = 202,  
2759 ["Euml"] = 203,  
2760 ["Igrave"] = 204,  
2761 ["Iacute"] = 205,  
2762 ["Icirc"] = 206,  
2763 ["Iuml"] = 207,  
2764 ["ETH"] = 208,  
2765 ["Ntilde"] = 209,  
2766 ["Ograve"] = 210,  
2767 ["Oacute"] = 211,  
2768 ["Ocirc"] = 212,  
2769 ["Otilde"] = 213,  
2770 ["Ouml"] = 214,  
2771 ["times"] = 215,  
2772 ["Oslash"] = 216,  
2773 ["Ugrave"] = 217,  
2774 ["Uacute"] = 218,  
2775 ["Ucirc"] = 219,  
2776 ["Uuml"] = 220,  
2777 ["Yacute"] = 221,  
2778 ["THORN"] = 222,  
2779 ["szlig"] = 223,  
2780 ["agrave"] = 224,  
2781 ["aacute"] = 225,  
2782 ["acirc"] = 226,  
2783 ["atilde"] = 227,  
2784 ["auml"] = 228,  
2785 ["aring"] = 229,  
2786 ["aelig"] = 230,  
2787 ["ccedil"] = 231,  
2788 ["egrave"] = 232,

2789 ["eacute"] = 233,  
2790 ["ecirc"] = 234,  
2791 ["euml"] = 235,  
2792 ["igrave"] = 236,  
2793 ["iacute"] = 237,  
2794 ["icirc"] = 238,  
2795 ["iuml"] = 239,  
2796 ["eth"] = 240,  
2797 ["ntilde"] = 241,  
2798 ["ograve"] = 242,  
2799 ["oacute"] = 243,  
2800 ["ocirc"] = 244,  
2801 ["otilde"] = 245,  
2802 ["ouml"] = 246,  
2803 ["divide"] = 247,  
2804 ["div"] = 247,  
2805 ["oslash"] = 248,  
2806 ["ugrave"] = 249,  
2807 ["uacute"] = 250,  
2808 ["ucirc"] = 251,  
2809 ["uuml"] = 252,  
2810 ["yacute"] = 253,  
2811 ["thorn"] = 254,  
2812 ["yuml"] = 255,  
2813 ["Amacr"] = 256,  
2814 ["amacr"] = 257,  
2815 ["Abreve"] = 258,  
2816 ["abreve"] = 259,  
2817 ["Aogon"] = 260,  
2818 ["aogon"] = 261,  
2819 ["Cacute"] = 262,  
2820 ["cacute"] = 263,  
2821 ["Ccirc"] = 264,  
2822 ["ccirc"] = 265,  
2823 ["Cdot"] = 266,  
2824 ["cdot"] = 267,  
2825 ["Ccaron"] = 268,  
2826 ["ccaron"] = 269,  
2827 ["Dcaron"] = 270,  
2828 ["dcaron"] = 271,  
2829 ["Dstrok"] = 272,  
2830 ["dstrok"] = 273,  
2831 ["Emacr"] = 274,  
2832 ["emacr"] = 275,  
2833 ["Edot"] = 278,  
2834 ["edot"] = 279,  
2835 ["Eogon"] = 280,

2836 ["eogon"] = 281,  
2837 ["Ecaron"] = 282,  
2838 ["ecaron"] = 283,  
2839 ["Gcirc"] = 284,  
2840 ["gcirc"] = 285,  
2841 ["Gbreve"] = 286,  
2842 ["gbreve"] = 287,  
2843 ["Gdot"] = 288,  
2844 ["gdot"] = 289,  
2845 ["Gcedil"] = 290,  
2846 ["Hcirc"] = 292,  
2847 ["hcirc"] = 293,  
2848 ["Hstrok"] = 294,  
2849 ["hstrok"] = 295,  
2850 ["Itilde"] = 296,  
2851 ["itilde"] = 297,  
2852 ["Imacr"] = 298,  
2853 ["imacr"] = 299,  
2854 ["Iogon"] = 302,  
2855 ["iogon"] = 303,  
2856 ["Idot"] = 304,  
2857 ["imath"] = 305,  
2858 ["inodot"] = 305,  
2859 ["IJlig"] = 306,  
2860 ["ijlig"] = 307,  
2861 ["Jcirc"] = 308,  
2862 ["jcirc"] = 309,  
2863 ["Kcedil"] = 310,  
2864 ["kcedil"] = 311,  
2865 ["kgreen"] = 312,  
2866 ["Lacute"] = 313,  
2867 ["lacute"] = 314,  
2868 ["Lcedil"] = 315,  
2869 ["lcedil"] = 316,  
2870 ["Lcaron"] = 317,  
2871 ["lcaron"] = 318,  
2872 ["Lmidot"] = 319,  
2873 ["lmidot"] = 320,  
2874 ["Lstrok"] = 321,  
2875 ["lstrok"] = 322,  
2876 ["Nacute"] = 323,  
2877 ["nacute"] = 324,  
2878 ["Ncedil"] = 325,  
2879 ["ncedil"] = 326,  
2880 ["Ncaron"] = 327,  
2881 ["ncaron"] = 328,  
2882 ["napos"] = 329,

2883 ["ENG"] = 330,  
2884 ["eng"] = 331,  
2885 ["Omacr"] = 332,  
2886 ["omacr"] = 333,  
2887 ["Odblac"] = 336,  
2888 ["odblac"] = 337,  
2889 ["OElig"] = 338,  
2890 ["oelig"] = 339,  
2891 ["Racute"] = 340,  
2892 ["racute"] = 341,  
2893 ["Rcedil"] = 342,  
2894 ["rcedil"] = 343,  
2895 ["Rcaron"] = 344,  
2896 ["rcaron"] = 345,  
2897 ["Sacute"] = 346,  
2898 ["sacute"] = 347,  
2899 ["Scirc"] = 348,  
2900 ["scirc"] = 349,  
2901 ["Scedil"] = 350,  
2902 ["scedil"] = 351,  
2903 ["Scaron"] = 352,  
2904 ["scaron"] = 353,  
2905 ["Tcedil"] = 354,  
2906 ["tcedil"] = 355,  
2907 ["Tcaron"] = 356,  
2908 ["tcaron"] = 357,  
2909 ["Tstrok"] = 358,  
2910 ["tstrok"] = 359,  
2911 ["Utilde"] = 360,  
2912 ["utilde"] = 361,  
2913 ["Umacr"] = 362,  
2914 ["umacr"] = 363,  
2915 ["Ubreve"] = 364,  
2916 ["ubreve"] = 365,  
2917 ["Uring"] = 366,  
2918 ["uring"] = 367,  
2919 ["Udblac"] = 368,  
2920 ["udblac"] = 369,  
2921 ["Uogon"] = 370,  
2922 ["uogon"] = 371,  
2923 ["Wcirc"] = 372,  
2924 ["wcirc"] = 373,  
2925 ["Ycirc"] = 374,  
2926 ["ycirc"] = 375,  
2927 ["Yuml"] = 376,  
2928 ["Zacute"] = 377,  
2929 ["zacute"] = 378,

2930 ["Zdot"] = 379,  
2931 ["zdot"] = 380,  
2932 ["Zcaron"] = 381,  
2933 ["zcaron"] = 382,  
2934 ["fnof"] = 402,  
2935 ["imped"] = 437,  
2936 ["gacute"] = 501,  
2937 ["jmath"] = 567,  
2938 ["circ"] = 710,  
2939 ["caron"] = 711,  
2940 ["Hacek"] = 711,  
2941 ["breve"] = 728,  
2942 ["Breve"] = 728,  
2943 ["dot"] = 729,  
2944 ["DiacriticalDot"] = 729,  
2945 ["ring"] = 730,  
2946 ["ogon"] = 731,  
2947 ["tilde"] = 732,  
2948 ["DiacriticalTilde"] = 732,  
2949 ["dblac"] = 733,  
2950 ["DiacriticalDoubleAcute"] = 733,  
2951 ["DownBreve"] = 785,  
2952 ["UnderBar"] = 818,  
2953 ["Alpha"] = 913,  
2954 ["Beta"] = 914,  
2955 ["Gamma"] = 915,  
2956 ["Delta"] = 916,  
2957 ["Epsilon"] = 917,  
2958 ["Zeta"] = 918,  
2959 ["Eta"] = 919,  
2960 ["Theta"] = 920,  
2961 ["Iota"] = 921,  
2962 ["Kappa"] = 922,  
2963 ["Lambda"] = 923,  
2964 ["Mu"] = 924,  
2965 ["Nu"] = 925,  
2966 ["Xi"] = 926,  
2967 ["Omicron"] = 927,  
2968 ["Pi"] = 928,  
2969 ["Rho"] = 929,  
2970 ["Sigma"] = 931,  
2971 ["Tau"] = 932,  
2972 ["Upsilon"] = 933,  
2973 ["Phi"] = 934,  
2974 ["Chi"] = 935,  
2975 ["Psi"] = 936,  
2976 ["Omega"] = 937,

2977 ["alpha"] = 945,  
2978 ["beta"] = 946,  
2979 ["gamma"] = 947,  
2980 ["delta"] = 948,  
2981 ["epsiv"] = 949,  
2982 ["varepsilon"] = 949,  
2983 ["epsilon"] = 949,  
2984 ["zeta"] = 950,  
2985 ["eta"] = 951,  
2986 ["theta"] = 952,  
2987 ["iota"] = 953,  
2988 ["kappa"] = 954,  
2989 ["lambda"] = 955,  
2990 ["mu"] = 956,  
2991 ["nu"] = 957,  
2992 ["xi"] = 958,  
2993 ["omicron"] = 959,  
2994 ["pi"] = 960,  
2995 ["rho"] = 961,  
2996 ["sigmav"] = 962,  
2997 ["varsigma"] = 962,  
2998 ["sigmaf"] = 962,  
2999 ["sigma"] = 963,  
3000 ["tau"] = 964,  
3001 ["upsilon"] = 965,  
3002 ["upsilon"] = 965,  
3003 ["phi"] = 966,  
3004 ["phiv"] = 966,  
3005 ["varphi"] = 966,  
3006 ["chi"] = 967,  
3007 ["psi"] = 968,  
3008 ["omega"] = 969,  
3009 ["thetav"] = 977,  
3010 ["vartheta"] = 977,  
3011 ["thetasym"] = 977,  
3012 ["Upsilon"] = 978,  
3013 ["upsih"] = 978,  
3014 ["straightphi"] = 981,  
3015 ["piv"] = 982,  
3016 ["varpi"] = 982,  
3017 ["Gammad"] = 988,  
3018 ["gammad"] = 989,  
3019 ["digamma"] = 989,  
3020 ["kappav"] = 1008,  
3021 ["varkappa"] = 1008,  
3022 ["rhov"] = 1009,  
3023 ["varrho"] = 1009,

3024 ["epsi"] = 1013,  
3025 ["straightepsilon"] = 1013,  
3026 ["bepsi"] = 1014,  
3027 ["backepsilon"] = 1014,  
3028 ["IOcy"] = 1025,  
3029 ["DJcy"] = 1026,  
3030 ["GJcy"] = 1027,  
3031 ["Jukcy"] = 1028,  
3032 ["DScy"] = 1029,  
3033 ["Iukcy"] = 1030,  
3034 ["YIcy"] = 1031,  
3035 ["Jsercy"] = 1032,  
3036 ["LJcy"] = 1033,  
3037 ["NJcy"] = 1034,  
3038 ["TSHcy"] = 1035,  
3039 ["KJcy"] = 1036,  
3040 ["Ubrcy"] = 1038,  
3041 ["DZcy"] = 1039,  
3042 ["Acy"] = 1040,  
3043 ["Bcy"] = 1041,  
3044 ["Vcy"] = 1042,  
3045 ["Gcy"] = 1043,  
3046 ["Dcy"] = 1044,  
3047 ["IEcy"] = 1045,  
3048 ["ZHcy"] = 1046,  
3049 ["Zcy"] = 1047,  
3050 ["Icy"] = 1048,  
3051 ["Jcy"] = 1049,  
3052 ["Kcy"] = 1050,  
3053 ["Lcy"] = 1051,  
3054 ["Mcy"] = 1052,  
3055 ["Ncy"] = 1053,  
3056 ["Ocy"] = 1054,  
3057 ["Pcy"] = 1055,  
3058 ["Rcy"] = 1056,  
3059 ["Scy"] = 1057,  
3060 ["Tcy"] = 1058,  
3061 ["Ucy"] = 1059,  
3062 ["Fcy"] = 1060,  
3063 ["KHcy"] = 1061,  
3064 ["TScy"] = 1062,  
3065 ["CHcy"] = 1063,  
3066 ["SHcy"] = 1064,  
3067 ["SHCHcy"] = 1065,  
3068 ["HARDcy"] = 1066,  
3069 ["Ycy"] = 1067,  
3070 ["SOFTcy"] = 1068,



3071 ["Ecy"] = 1069,  
3072 ["YUcy"] = 1070,  
3073 ["YAcy"] = 1071,  
3074 ["acy"] = 1072,  
3075 ["bcy"] = 1073,  
3076 ["vcy"] = 1074,  
3077 ["gcy"] = 1075,  
3078 ["dcy"] = 1076,  
3079 ["iecy"] = 1077,  
3080 ["zhcy"] = 1078,  
3081 ["zcy"] = 1079,  
3082 ["icy"] = 1080,  
3083 ["jcy"] = 1081,  
3084 ["kcy"] = 1082,  
3085 ["lcy"] = 1083,  
3086 ["mcy"] = 1084,  
3087 ["ncy"] = 1085,  
3088 ["ocy"] = 1086,  
3089 ["pcy"] = 1087,  
3090 ["rcy"] = 1088,  
3091 ["scy"] = 1089,  
3092 ["tcy"] = 1090,  
3093 ["ucy"] = 1091,  
3094 ["fcy"] = 1092,  
3095 ["khcy"] = 1093,  
3096 ["tscy"] = 1094,  
3097 ["chcy"] = 1095,  
3098 ["shcy"] = 1096,  
3099 ["shchcy"] = 1097,  
3100 ["hardcy"] = 1098,  
3101 ["ycy"] = 1099,  
3102 ["softcy"] = 1100,  
3103 ["ecy"] = 1101,  
3104 ["yucy"] = 1102,  
3105 ["yacy"] = 1103,  
3106 ["iocy"] = 1105,  
3107 ["djcy"] = 1106,  
3108 ["gjcy"] = 1107,  
3109 ["jukcy"] = 1108,  
3110 ["dscy"] = 1109,  
3111 ["iukcy"] = 1110,  
3112 ["yicy"] = 1111,  
3113 ["jsercy"] = 1112,  
3114 ["ljcy"] = 1113,  
3115 ["njcy"] = 1114,  
3116 ["tshcy"] = 1115,  
3117 ["kjcy"] = 1116,

3118 ["ubrscy"] = 1118,  
3119 ["dzcycy"] = 1119,  
3120 ["ensp"] = 8194,  
3121 ["emsp"] = 8195,  
3122 ["emsp13"] = 8196,  
3123 ["emsp14"] = 8197,  
3124 ["numsp"] = 8199,  
3125 ["puncsp"] = 8200,  
3126 ["thinsp"] = 8201,  
3127 ["ThinSpace"] = 8201,  
3128 ["hairsp"] = 8202,  
3129 ["VeryThinSpace"] = 8202,  
3130 ["ZeroWidthSpace"] = 8203,  
3131 ["NegativeVeryThinSpace"] = 8203,  
3132 ["NegativeThinSpace"] = 8203,  
3133 ["NegativeMediumSpace"] = 8203,  
3134 ["NegativeThickSpace"] = 8203,  
3135 ["zwnj"] = 8204,  
3136 ["zwj"] = 8205,  
3137 ["lrm"] = 8206,  
3138 ["rlm"] = 8207,  
3139 ["hyphen"] = 8208,  
3140 ["dash"] = 8208,  
3141 ["ndash"] = 8211,  
3142 ["mdash"] = 8212,  
3143 ["horbar"] = 8213,  
3144 ["Verbar"] = 8214,  
3145 ["Vert"] = 8214,  
3146 ["lsquo"] = 8216,  
3147 ["OpenCurlyQuote"] = 8216,  
3148 ["rsquo"] = 8217,  
3149 ["rsquor"] = 8217,  
3150 ["CloseCurlyQuote"] = 8217,  
3151 ["lsquor"] = 8218,  
3152 ["sbquo"] = 8218,  
3153 ["ldquo"] = 8220,  
3154 ["OpenCurlyDoubleQuote"] = 8220,  
3155 ["rdquo"] = 8221,  
3156 ["rdquor"] = 8221,  
3157 ["CloseCurlyDoubleQuote"] = 8221,  
3158 ["ldquor"] = 8222,  
3159 ["bdquo"] = 8222,  
3160 ["dagger"] = 8224,  
3161 ["Dagger"] = 8225,  
3162 ["ddagger"] = 8225,  
3163 ["bull"] = 8226,  
3164 ["bullet"] = 8226,

3165 ["nldr"] = 8229,  
3166 ["hellip"] = 8230,  
3167 ["mldr"] = 8230,  
3168 ["permil"] = 8240,  
3169 ["pertenk"] = 8241,  
3170 ["prime"] = 8242,  
3171 ["Prime"] = 8243,  
3172 ["tprime"] = 8244,  
3173 ["bprime"] = 8245,  
3174 ["backprime"] = 8245,  
3175 ["lsaquo"] = 8249,  
3176 ["rsaquo"] = 8250,  
3177 ["oline"] = 8254,  
3178 ["caret"] = 8257,  
3179 ["hybull"] = 8259,  
3180 ["frasl"] = 8260,  
3181 ["bsemi"] = 8271,  
3182 ["qprime"] = 8279,  
3183 ["MediumSpace"] = 8287,  
3184 ["NoBreak"] = 8288,  
3185 ["ApplyFunction"] = 8289,  
3186 ["af"] = 8289,  
3187 ["InvisibleTimes"] = 8290,  
3188 ["it"] = 8290,  
3189 ["InvisibleComma"] = 8291,  
3190 ["ic"] = 8291,  
3191 ["euro"] = 8364,  
3192 ["tdot"] = 8411,  
3193 ["TripleDot"] = 8411,  
3194 ["DotDot"] = 8412,  
3195 ["Copf"] = 8450,  
3196 ["complexes"] = 8450,  
3197 ["incare"] = 8453,  
3198 ["gscr"] = 8458,  
3199 ["hamilt"] = 8459,  
3200 ["HilbertSpace"] = 8459,  
3201 ["Hscr"] = 8459,  
3202 ["Hfr"] = 8460,  
3203 ["Poincareplane"] = 8460,  
3204 ["quaternions"] = 8461,  
3205 ["Hopf"] = 8461,  
3206 ["planckh"] = 8462,  
3207 ["planck"] = 8463,  
3208 ["hbar"] = 8463,  
3209 ["plankv"] = 8463,  
3210 ["hslash"] = 8463,  
3211 ["Iscr"] = 8464,

3212 ["imagline"] = 8464,  
3213 ["image"] = 8465,  
3214 ["Im"] = 8465,  
3215 ["imagpart"] = 8465,  
3216 ["Ifr"] = 8465,  
3217 ["Lscr"] = 8466,  
3218 ["lagran"] = 8466,  
3219 ["Laplacetrif"] = 8466,  
3220 ["ell"] = 8467,  
3221 ["Nopf"] = 8469,  
3222 ["naturals"] = 8469,  
3223 ["numero"] = 8470,  
3224 ["copysr"] = 8471,  
3225 ["weierp"] = 8472,  
3226 ["wp"] = 8472,  
3227 ["Popf"] = 8473,  
3228 ["primes"] = 8473,  
3229 ["rationals"] = 8474,  
3230 ["Qopf"] = 8474,  
3231 ["Rscr"] = 8475,  
3232 ["realine"] = 8475,  
3233 ["real"] = 8476,  
3234 ["Re"] = 8476,  
3235 ["realpart"] = 8476,  
3236 ["Rfr"] = 8476,  
3237 ["reals"] = 8477,  
3238 ["Ropf"] = 8477,  
3239 ["rx"] = 8478,  
3240 ["trade"] = 8482,  
3241 ["TRADE"] = 8482,  
3242 ["integers"] = 8484,  
3243 ["Zopf"] = 8484,  
3244 ["ohm"] = 8486,  
3245 ["mho"] = 8487,  
3246 ["Zfr"] = 8488,  
3247 ["zeetrif"] = 8488,  
3248 ["iiota"] = 8489,  
3249 ["angst"] = 8491,  
3250 ["bernou"] = 8492,  
3251 ["Bernoullis"] = 8492,  
3252 ["Bscr"] = 8492,  
3253 ["Cfr"] = 8493,  
3254 ["Cayleys"] = 8493,  
3255 ["escr"] = 8495,  
3256 ["Escr"] = 8496,  
3257 ["expectation"] = 8496,  
3258 ["Fscr"] = 8497,

3259 ["Fouriertrf"] = 8497,  
 3260 ["phmmat"] = 8499,  
 3261 ["Mellintrf"] = 8499,  
 3262 ["Mscr"] = 8499,  
 3263 ["order"] = 8500,  
 3264 ["orderof"] = 8500,  
 3265 ["oscr"] = 8500,  
 3266 ["alefsym"] = 8501,  
 3267 ["aleph"] = 8501,  
 3268 ["beth"] = 8502,  
 3269 ["gimel"] = 8503,  
 3270 ["daleth"] = 8504,  
 3271 ["CapitalDifferentialD"] = 8517,  
 3272 ["DD"] = 8517,  
 3273 ["DifferentialD"] = 8518,  
 3274 ["dd"] = 8518,  
 3275 ["ExponentialE"] = 8519,  
 3276 ["exponentiale"] = 8519,  
 3277 ["ee"] = 8519,  
 3278 ["ImaginaryI"] = 8520,  
 3279 ["ii"] = 8520,  
 3280 ["frac13"] = 8531,  
 3281 ["frac23"] = 8532,  
 3282 ["frac15"] = 8533,  
 3283 ["frac25"] = 8534,  
 3284 ["frac35"] = 8535,  
 3285 ["frac45"] = 8536,  
 3286 ["frac16"] = 8537,  
 3287 ["frac56"] = 8538,  
 3288 ["frac18"] = 8539,  
 3289 ["frac38"] = 8540,  
 3290 ["frac58"] = 8541,  
 3291 ["frac78"] = 8542,  
 3292 ["larr"] = 8592,  
 3293 ["leftarrow"] = 8592,  
 3294 ["LeftArrow"] = 8592,  
 3295 ["slarr"] = 8592,  
 3296 ["ShortLeftArrow"] = 8592,  
 3297 ["uarr"] = 8593,  
 3298 ["uparrow"] = 8593,  
 3299 ["UpArrow"] = 8593,  
 3300 ["ShortUpArrow"] = 8593,  
 3301 ["rarr"] = 8594,  
 3302 ["rightarrow"] = 8594,  
 3303 ["RightArrow"] = 8594,  
 3304 ["srarr"] = 8594,  
 3305 ["ShortRightArrow"] = 8594,

```

3306 ["darr"] = 8595,
3307 ["downarrow"] = 8595,
3308 ["DownArrow"] = 8595,
3309 ["ShortDownArrow"] = 8595,
3310 ["harr"] = 8596,
3311 ["leftrightarrow"] = 8596,
3312 ["LeftRightArrow"] = 8596,
3313 ["varr"] = 8597,
3314 ["updownarrow"] = 8597,
3315 ["UpDownArrow"] = 8597,
3316 ["nwarr"] = 8598,
3317 ["UpperLeftArrow"] = 8598,
3318 ["nwarrow"] = 8598,
3319 ["nearr"] = 8599,
3320 ["UpperRightArrow"] = 8599,
3321 ["nearrow"] = 8599,
3322 ["searr"] = 8600,
3323 ["searrow"] = 8600,
3324 ["LowerRightArrow"] = 8600,
3325 ["swarr"] = 8601,
3326 ["swarrow"] = 8601,
3327 ["LowerLeftArrow"] = 8601,
3328 ["nlarr"] = 8602,
3329 ["nleftarrow"] = 8602,
3330 ["nrarr"] = 8603,
3331 ["nrightarrow"] = 8603,
3332 ["rarrw"] = 8605,
3333 ["rightsquigarrow"] = 8605,
3334 ["Larr"] = 8606,
3335 ["twoheadleftarrow"] = 8606,
3336 ["Uarr"] = 8607,
3337 ["Rarr"] = 8608,
3338 ["twoheadrightarrow"] = 8608,
3339 ["Darr"] = 8609,
3340 ["larrtl"] = 8610,
3341 ["leftarrowtail"] = 8610,
3342 ["rarrtl"] = 8611,
3343 ["rightarrowtail"] = 8611,
3344 ["LeftTeeArrow"] = 8612,
3345 ["mapstoleft"] = 8612,
3346 ["UpTeeArrow"] = 8613,
3347 ["mapstoup"] = 8613,
3348 ["map"] = 8614,
3349 ["RightTeeArrow"] = 8614,
3350 ["mapsto"] = 8614,
3351 ["DownTeeArrow"] = 8615,
3352 ["mapstodown"] = 8615,

```

3353 ["larrhk"] = 8617,  
 3354 ["hookleftarrow"] = 8617,  
 3355 ["rarrhk"] = 8618,  
 3356 ["hookrightarrow"] = 8618,  
 3357 ["larrlp"] = 8619,  
 3358 ["looparrowleft"] = 8619,  
 3359 ["rarrlp"] = 8620,  
 3360 ["looparrowright"] = 8620,  
 3361 ["harrw"] = 8621,  
 3362 ["leftrightsquigarrow"] = 8621,  
 3363 ["nharr"] = 8622,  
 3364 ["nletrightarrow"] = 8622,  
 3365 ["lsh"] = 8624,  
 3366 ["Lsh"] = 8624,  
 3367 ["rsh"] = 8625,  
 3368 ["Rsh"] = 8625,  
 3369 ["ldsh"] = 8626,  
 3370 ["rdsh"] = 8627,  
 3371 ["crarr"] = 8629,  
 3372 ["cularr"] = 8630,  
 3373 ["curvearrowleft"] = 8630,  
 3374 ["curarr"] = 8631,  
 3375 ["curvearrowright"] = 8631,  
 3376 ["olarr"] = 8634,  
 3377 ["circlearrowleft"] = 8634,  
 3378 ["orarr"] = 8635,  
 3379 ["circlearrowright"] = 8635,  
 3380 ["lharu"] = 8636,  
 3381 ["LeftVector"] = 8636,  
 3382 ["leftharpoonup"] = 8636,  
 3383 ["lhard"] = 8637,  
 3384 ["leftharpoondown"] = 8637,  
 3385 ["DownLeftVector"] = 8637,  
 3386 ["uharr"] = 8638,  
 3387 ["upharpoonright"] = 8638,  
 3388 ["RightUpVector"] = 8638,  
 3389 ["uharl"] = 8639,  
 3390 ["upharpoonleft"] = 8639,  
 3391 ["LeftUpVector"] = 8639,  
 3392 ["rharu"] = 8640,  
 3393 ["RightVector"] = 8640,  
 3394 ["rightharpoonup"] = 8640,  
 3395 ["rhard"] = 8641,  
 3396 ["rightharpoondown"] = 8641,  
 3397 ["DownRightVector"] = 8641,  
 3398 ["dharr"] = 8642,  
 3399 ["RightDownVector"] = 8642,

3400 ["downharpoonright"] = 8642,  
 3401 ["dharl"] = 8643,  
 3402 ["LeftDownVector"] = 8643,  
 3403 ["downharpoonleft"] = 8643,  
 3404 ["rlarr"] = 8644,  
 3405 ["rightleftarrows"] = 8644,  
 3406 ["RightArrowLeftArrow"] = 8644,  
 3407 ["udarr"] = 8645,  
 3408 ["UpArrowDownArrow"] = 8645,  
 3409 ["lrarr"] = 8646,  
 3410 ["leftrightarrows"] = 8646,  
 3411 ["LeftArrowRightArrow"] = 8646,  
 3412 ["llarr"] = 8647,  
 3413 ["leftleftarrows"] = 8647,  
 3414 ["uuarr"] = 8648,  
 3415 ["upuparrows"] = 8648,  
 3416 ["rrarr"] = 8649,  
 3417 ["rightrightarrows"] = 8649,  
 3418 ["ddarr"] = 8650,  
 3419 ["downdownarrows"] = 8650,  
 3420 ["lrhar"] = 8651,  
 3421 ["ReverseEquilibrium"] = 8651,  
 3422 ["leftrightharpoons"] = 8651,  
 3423 ["rlhar"] = 8652,  
 3424 ["rightleftharpoons"] = 8652,  
 3425 ["Equilibrium"] = 8652,  
 3426 ["nlArr"] = 8653,  
 3427 ["nLeftarrow"] = 8653,  
 3428 ["nhArr"] = 8654,  
 3429 ["nLeftrightarrow"] = 8654,  
 3430 ["nrArr"] = 8655,  
 3431 ["nRightarrow"] = 8655,  
 3432 ["lArr"] = 8656,  
 3433 ["Leftarrow"] = 8656,  
 3434 ["DoubleLeftArrow"] = 8656,  
 3435 ["uArr"] = 8657,  
 3436 ["Uparrow"] = 8657,  
 3437 ["DoubleUpArrow"] = 8657,  
 3438 ["rArr"] = 8658,  
 3439 ["Rightarrow"] = 8658,  
 3440 ["Implies"] = 8658,  
 3441 ["DoubleRightArrow"] = 8658,  
 3442 ["dArr"] = 8659,  
 3443 ["Downarrow"] = 8659,  
 3444 ["DoubleDownArrow"] = 8659,  
 3445 ["hArr"] = 8660,  
 3446 ["Leftrightarrow"] = 8660,



3447 ["DoubleLeftRightArrow"] = 8660,  
3448 ["iff"] = 8660,  
3449 ["vArr"] = 8661,  
3450 ["Updownarrow"] = 8661,  
3451 ["DoubleUpDownArrow"] = 8661,  
3452 ["nwArr"] = 8662,  
3453 ["neArr"] = 8663,  
3454 ["seArr"] = 8664,  
3455 ["swArr"] = 8665,  
3456 ["lAarr"] = 8666,  
3457 ["Lleftarrow"] = 8666,  
3458 ["rAarr"] = 8667,  
3459 ["Rrightarrow"] = 8667,  
3460 ["zigrarr"] = 8669,  
3461 ["larrb"] = 8676,  
3462 ["LeftArrowBar"] = 8676,  
3463 ["rarrb"] = 8677,  
3464 ["RightArrowBar"] = 8677,  
3465 ["duarr"] = 8693,  
3466 ["DownArrowUpArrow"] = 8693,  
3467 ["loarr"] = 8701,  
3468 ["roarr"] = 8702,  
3469 ["hoarr"] = 8703,  
3470 ["forall"] = 8704,  
3471 ["ForAll"] = 8704,  
3472 ["comp"] = 8705,  
3473 ["complement"] = 8705,  
3474 ["part"] = 8706,  
3475 ["PartialD"] = 8706,  
3476 ["exist"] = 8707,  
3477 ["Exists"] = 8707,  
3478 ["nexist"] = 8708,  
3479 ["NotExists"] = 8708,  
3480 ["nexists"] = 8708,  
3481 ["empty"] = 8709,  
3482 ["emptyset"] = 8709,  
3483 ["emptyv"] = 8709,  
3484 ["varnothing"] = 8709,  
3485 ["nabla"] = 8711,  
3486 ["Del"] = 8711,  
3487 ["isin"] = 8712,  
3488 ["isinv"] = 8712,  
3489 ["Element"] = 8712,  
3490 ["in"] = 8712,  
3491 ["notin"] = 8713,  
3492 ["NotElement"] = 8713,  
3493 ["notinva"] = 8713,

3494 ["niv"] = 8715,  
 3495 ["ReverseElement"] = 8715,  
 3496 ["ni"] = 8715,  
 3497 ["SuchThat"] = 8715,  
 3498 ["notni"] = 8716,  
 3499 ["notniva"] = 8716,  
 3500 ["NotReverseElement"] = 8716,  
 3501 ["prod"] = 8719,  
 3502 ["Product"] = 8719,  
 3503 ["coprod"] = 8720,  
 3504 ["Coproduct"] = 8720,  
 3505 ["sum"] = 8721,  
 3506 ["Sum"] = 8721,  
 3507 ["minus"] = 8722,  
 3508 ["mnplus"] = 8723,  
 3509 ["mp"] = 8723,  
 3510 ["MinusPlus"] = 8723,  
 3511 ["plusdo"] = 8724,  
 3512 ["dotplus"] = 8724,  
 3513 ["setmn"] = 8726,  
 3514 ["setminus"] = 8726,  
 3515 ["Backslash"] = 8726,  
 3516 ["ssetmn"] = 8726,  
 3517 ["smallsetminus"] = 8726,  
 3518 ["lowast"] = 8727,  
 3519 ["compfn"] = 8728,  
 3520 ["SmallCircle"] = 8728,  
 3521 ["radic"] = 8730,  
 3522 ["Sqrt"] = 8730,  
 3523 ["prop"] = 8733,  
 3524 ["propto"] = 8733,  
 3525 ["Proportional"] = 8733,  
 3526 ["vprop"] = 8733,  
 3527 ["varpropto"] = 8733,  
 3528 ["infin"] = 8734,  
 3529 ["angrt"] = 8735,  
 3530 ["ang"] = 8736,  
 3531 ["angle"] = 8736,  
 3532 ["angmsd"] = 8737,  
 3533 ["measuredangle"] = 8737,  
 3534 ["angsph"] = 8738,  
 3535 ["mid"] = 8739,  
 3536 ["VerticalBar"] = 8739,  
 3537 ["smid"] = 8739,  
 3538 ["shortmid"] = 8739,  
 3539 ["nmid"] = 8740,  
 3540 ["NotVerticalBar"] = 8740,

3541 ["nsmid"] = 8740,  
3542 ["nshortmid"] = 8740,  
3543 ["par"] = 8741,  
3544 ["parallel"] = 8741,  
3545 ["DoubleVerticalBar"] = 8741,  
3546 ["spar"] = 8741,  
3547 ["shortparallel"] = 8741,  
3548 ["npar"] = 8742,  
3549 ["nparallel"] = 8742,  
3550 ["NotDoubleVerticalBar"] = 8742,  
3551 ["nspar"] = 8742,  
3552 ["nshortparallel"] = 8742,  
3553 ["and"] = 8743,  
3554 ["wedge"] = 8743,  
3555 ["or"] = 8744,  
3556 ["vee"] = 8744,  
3557 ["cap"] = 8745,  
3558 ["cup"] = 8746,  
3559 ["int"] = 8747,  
3560 ["Integral"] = 8747,  
3561 ["Int"] = 8748,  
3562 ["tint"] = 8749,  
3563 ["iiint"] = 8749,  
3564 ["conint"] = 8750,  
3565 ["oint"] = 8750,  
3566 ["ContourIntegral"] = 8750,  
3567 ["Conint"] = 8751,  
3568 ["DoubleContourIntegral"] = 8751,  
3569 ["Cconint"] = 8752,  
3570 ["cwint"] = 8753,  
3571 ["cwconint"] = 8754,  
3572 ["ClockwiseContourIntegral"] = 8754,  
3573 ["awconint"] = 8755,  
3574 ["CounterClockwiseContourIntegral"] = 8755,  
3575 ["there4"] = 8756,  
3576 ["therefore"] = 8756,  
3577 ["Therefore"] = 8756,  
3578 ["because"] = 8757,  
3579 ["because"] = 8757,  
3580 ["Because"] = 8757,  
3581 ["ratio"] = 8758,  
3582 ["Colon"] = 8759,  
3583 ["Proportion"] = 8759,  
3584 ["minusd"] = 8760,  
3585 ["dotminus"] = 8760,  
3586 ["mDDot"] = 8762,  
3587 ["homtht"] = 8763,

3588 ["sim"] = 8764,  
3589 ["Tilde"] = 8764,  
3590 ["thksim"] = 8764,  
3591 ["thicksim"] = 8764,  
3592 ["bsim"] = 8765,  
3593 ["backsim"] = 8765,  
3594 ["ac"] = 8766,  
3595 ["mstpos"] = 8766,  
3596 ["acd"] = 8767,  
3597 ["wreath"] = 8768,  
3598 ["VerticalTilde"] = 8768,  
3599 ["wr"] = 8768,  
3600 ["nsim"] = 8769,  
3601 ["NotTilde"] = 8769,  
3602 ["esim"] = 8770,  
3603 ["EqualTilde"] = 8770,  
3604 ["eqsim"] = 8770,  
3605 ["sime"] = 8771,  
3606 ["TildeEqual"] = 8771,  
3607 ["simeq"] = 8771,  
3608 ["nsime"] = 8772,  
3609 ["nsimeq"] = 8772,  
3610 ["NotTildeEqual"] = 8772,  
3611 ["cong"] = 8773,  
3612 ["TildeFullEqual"] = 8773,  
3613 ["simne"] = 8774,  
3614 ["ncong"] = 8775,  
3615 ["NotTildeFullEqual"] = 8775,  
3616 ["asympt"] = 8776,  
3617 ["ap"] = 8776,  
3618 ["TildeTilde"] = 8776,  
3619 ["approx"] = 8776,  
3620 ["thkap"] = 8776,  
3621 ["thickapprox"] = 8776,  
3622 ["nap"] = 8777,  
3623 ["NotTildeTilde"] = 8777,  
3624 ["napprox"] = 8777,  
3625 ["ape"] = 8778,  
3626 ["approxpeq"] = 8778,  
3627 ["apid"] = 8779,  
3628 ["bcong"] = 8780,  
3629 ["backcong"] = 8780,  
3630 ["asympeq"] = 8781,  
3631 ["CupCap"] = 8781,  
3632 ["bump"] = 8782,  
3633 ["HumpDownHump"] = 8782,  
3634 ["Bumpeq"] = 8782,

3635 ["bump\_e"] = 8783,  
 3636 ["HumpEqual"] = 8783,  
 3637 ["bump\_eq"] = 8783,  
 3638 ["esdot"] = 8784,  
 3639 ["DotEqual"] = 8784,  
 3640 ["doteq"] = 8784,  
 3641 ["eDot"] = 8785,  
 3642 ["doteqdot"] = 8785,  
 3643 ["efDot"] = 8786,  
 3644 ["fallingdotseq"] = 8786,  
 3645 ["erDot"] = 8787,  
 3646 ["risingdotseq"] = 8787,  
 3647 ["colone"] = 8788,  
 3648 ["coloneq"] = 8788,  
 3649 ["Assign"] = 8788,  
 3650 ["ecolon"] = 8789,  
 3651 ["eqcolon"] = 8789,  
 3652 ["ecir"] = 8790,  
 3653 ["eqcirc"] = 8790,  
 3654 ["cire"] = 8791,  
 3655 ["circeq"] = 8791,  
 3656 ["wedgeq"] = 8793,  
 3657 ["veeeq"] = 8794,  
 3658 ["trie"] = 8796,  
 3659 ["triangleq"] = 8796,  
 3660 ["equest"] = 8799,  
 3661 ["questeq"] = 8799,  
 3662 ["ne"] = 8800,  
 3663 ["NotEqual"] = 8800,  
 3664 ["equiv"] = 8801,  
 3665 ["Congruent"] = 8801,  
 3666 ["nequiv"] = 8802,  
 3667 ["NotCongruent"] = 8802,  
 3668 ["le"] = 8804,  
 3669 ["leq"] = 8804,  
 3670 ["ge"] = 8805,  
 3671 ["GreaterEqual"] = 8805,  
 3672 ["geq"] = 8805,  
 3673 ["lE"] = 8806,  
 3674 ["LessFullEqual"] = 8806,  
 3675 ["leqq"] = 8806,  
 3676 ["gE"] = 8807,  
 3677 ["GreaterFullEqual"] = 8807,  
 3678 ["geqq"] = 8807,  
 3679 ["lnE"] = 8808,  
 3680 ["lneqq"] = 8808,  
 3681 ["gnE"] = 8809,

3682 ["gneqq"] = 8809,  
 3683 ["Lt"] = 8810,  
 3684 ["NestedLessLess"] = 8810,  
 3685 ["ll"] = 8810,  
 3686 ["Gt"] = 8811,  
 3687 ["NestedGreaterGreater"] = 8811,  
 3688 ["gg"] = 8811,  
 3689 ["twixt"] = 8812,  
 3690 ["between"] = 8812,  
 3691 ["NotCupCap"] = 8813,  
 3692 ["nlt"] = 8814,  
 3693 ["NotLess"] = 8814,  
 3694 ["nless"] = 8814,  
 3695 ["ngt"] = 8815,  
 3696 ["NotGreater"] = 8815,  
 3697 ["ngtr"] = 8815,  
 3698 ["nle"] = 8816,  
 3699 ["NotLessEqual"] = 8816,  
 3700 ["nleq"] = 8816,  
 3701 ["nge"] = 8817,  
 3702 ["NotGreaterEqual"] = 8817,  
 3703 ["ngeq"] = 8817,  
 3704 ["lsim"] = 8818,  
 3705 ["LessTilde"] = 8818,  
 3706 ["lesssim"] = 8818,  
 3707 ["gsim"] = 8819,  
 3708 ["gtrsim"] = 8819,  
 3709 ["GreaterTilde"] = 8819,  
 3710 ["nlsim"] = 8820,  
 3711 ["NotLessTilde"] = 8820,  
 3712 ["ngsim"] = 8821,  
 3713 ["NotGreaterTilde"] = 8821,  
 3714 ["lg"] = 8822,  
 3715 ["lessgtr"] = 8822,  
 3716 ["LessGreater"] = 8822,  
 3717 ["gl"] = 8823,  
 3718 ["gtrless"] = 8823,  
 3719 ["GreaterLess"] = 8823,  
 3720 ["ntlg"] = 8824,  
 3721 ["NotLessGreater"] = 8824,  
 3722 ["ntgl"] = 8825,  
 3723 ["NotGreaterLess"] = 8825,  
 3724 ["pr"] = 8826,  
 3725 ["Precedes"] = 8826,  
 3726 ["prec"] = 8826,  
 3727 ["sc"] = 8827,  
 3728 ["Succeeds"] = 8827,

3729 ["succ"] = 8827,  
 3730 ["prcue"] = 8828,  
 3731 ["PrecedesSlantEqual"] = 8828,  
 3732 ["preccurlyeq"] = 8828,  
 3733 ["sccue"] = 8829,  
 3734 ["SucceedsSlantEqual"] = 8829,  
 3735 ["succurlyeq"] = 8829,  
 3736 ["prsim"] = 8830,  
 3737 ["precsim"] = 8830,  
 3738 ["PrecedesTilde"] = 8830,  
 3739 ["scsim"] = 8831,  
 3740 ["succsim"] = 8831,  
 3741 ["SucceedsTilde"] = 8831,  
 3742 ["npr"] = 8832,  
 3743 ["nprec"] = 8832,  
 3744 ["NotPrecedes"] = 8832,  
 3745 ["nsc"] = 8833,  
 3746 ["nsucc"] = 8833,  
 3747 ["NotSucceeds"] = 8833,  
 3748 ["sub"] = 8834,  
 3749 ["subset"] = 8834,  
 3750 ["sup"] = 8835,  
 3751 ["supset"] = 8835,  
 3752 ["Superset"] = 8835,  
 3753 ["nsub"] = 8836,  
 3754 ["nsup"] = 8837,  
 3755 ["sube"] = 8838,  
 3756 ["SubsetEqual"] = 8838,  
 3757 ["subseteq"] = 8838,  
 3758 ["supe"] = 8839,  
 3759 ["supseteq"] = 8839,  
 3760 ["SupersetEqual"] = 8839,  
 3761 ["nsube"] = 8840,  
 3762 ["nsubseteq"] = 8840,  
 3763 ["NotSubsetEqual"] = 8840,  
 3764 ["nsupe"] = 8841,  
 3765 ["nsupseteq"] = 8841,  
 3766 ["NotSupersetEqual"] = 8841,  
 3767 ["subne"] = 8842,  
 3768 ["subsetneq"] = 8842,  
 3769 ["supne"] = 8843,  
 3770 ["supsetneq"] = 8843,  
 3771 ["cupdot"] = 8845,  
 3772 ["uplus"] = 8846,  
 3773 ["UnionPlus"] = 8846,  
 3774 ["sqsub"] = 8847,  
 3775 ["SquareSubset"] = 8847,

3776 ["sqsubset"] = 8847,  
 3777 ["sqsup"] = 8848,  
 3778 ["SquareSuperset"] = 8848,  
 3779 ["sqsupset"] = 8848,  
 3780 ["sqsube"] = 8849,  
 3781 ["SquareSubsetEqual"] = 8849,  
 3782 ["sqsubseteq"] = 8849,  
 3783 ["sqsupe"] = 8850,  
 3784 ["SquareSupersetEqual"] = 8850,  
 3785 ["sqsupseteq"] = 8850,  
 3786 ["sqcap"] = 8851,  
 3787 ["SquareIntersection"] = 8851,  
 3788 ["sqcup"] = 8852,  
 3789 ["SquareUnion"] = 8852,  
 3790 ["oplus"] = 8853,  
 3791 ["CirclePlus"] = 8853,  
 3792 ["ominus"] = 8854,  
 3793 ["CircleMinus"] = 8854,  
 3794 ["otimes"] = 8855,  
 3795 ["CircleTimes"] = 8855,  
 3796 ["osol"] = 8856,  
 3797 ["odot"] = 8857,  
 3798 ["CircleDot"] = 8857,  
 3799 ["ocir"] = 8858,  
 3800 ["circledcirc"] = 8858,  
 3801 ["oast"] = 8859,  
 3802 ["circledast"] = 8859,  
 3803 ["odash"] = 8861,  
 3804 ["circleddash"] = 8861,  
 3805 ["plusb"] = 8862,  
 3806 ["boxplus"] = 8862,  
 3807 ["minusb"] = 8863,  
 3808 ["boxminus"] = 8863,  
 3809 ["timesb"] = 8864,  
 3810 ["boxtimes"] = 8864,  
 3811 ["sdotb"] = 8865,  
 3812 ["dotsquare"] = 8865,  
 3813 ["vdash"] = 8866,  
 3814 ["RightTee"] = 8866,  
 3815 ["dashv"] = 8867,  
 3816 ["LeftTee"] = 8867,  
 3817 ["top"] = 8868,  
 3818 ["DownTee"] = 8868,  
 3819 ["bottom"] = 8869,  
 3820 ["bot"] = 8869,  
 3821 ["perp"] = 8869,  
 3822 ["UpTee"] = 8869,



3823 ["models"] = 8871,  
 3824 ["vDash"] = 8872,  
 3825 ["DoubleRightTee"] = 8872,  
 3826 ["Vdash"] = 8873,  
 3827 ["Vvdash"] = 8874,  
 3828 ["VDash"] = 8875,  
 3829 ["nvdash"] = 8876,  
 3830 ["nvDash"] = 8877,  
 3831 ["nVdash"] = 8878,  
 3832 ["nVDash"] = 8879,  
 3833 ["prurel"] = 8880,  
 3834 ["vltri"] = 8882,  
 3835 ["vartriangleleft"] = 8882,  
 3836 ["LeftTriangle"] = 8882,  
 3837 ["vrtri"] = 8883,  
 3838 ["vartriangleright"] = 8883,  
 3839 ["RightTriangle"] = 8883,  
 3840 ["ltrie"] = 8884,  
 3841 ["trianglelefteq"] = 8884,  
 3842 ["LeftTriangleEqual"] = 8884,  
 3843 ["rtrie"] = 8885,  
 3844 ["trianglerighteq"] = 8885,  
 3845 ["RightTriangleEqual"] = 8885,  
 3846 ["origof"] = 8886,  
 3847 ["imof"] = 8887,  
 3848 ["mumap"] = 8888,  
 3849 ["multimap"] = 8888,  
 3850 ["hercon"] = 8889,  
 3851 ["intcal"] = 8890,  
 3852 ["intercal"] = 8890,  
 3853 ["veebar"] = 8891,  
 3854 ["barvee"] = 8893,  
 3855 ["angrtvb"] = 8894,  
 3856 ["lrtri"] = 8895,  
 3857 ["xwedge"] = 8896,  
 3858 ["Wedge"] = 8896,  
 3859 ["bigwedge"] = 8896,  
 3860 ["xvee"] = 8897,  
 3861 ["Vee"] = 8897,  
 3862 ["bigvee"] = 8897,  
 3863 ["xcap"] = 8898,  
 3864 ["Intersection"] = 8898,  
 3865 ["bigcap"] = 8898,  
 3866 ["xcup"] = 8899,  
 3867 ["Union"] = 8899,  
 3868 ["bigcup"] = 8899,  
 3869 ["diam"] = 8900,

3870 ["diamond"] = 8900,  
 3871 ["Diamond"] = 8900,  
 3872 ["sdot"] = 8901,  
 3873 ["sstarf"] = 8902,  
 3874 ["Star"] = 8902,  
 3875 ["divonx"] = 8903,  
 3876 ["divideontimes"] = 8903,  
 3877 ["bowtie"] = 8904,  
 3878 ["ltimes"] = 8905,  
 3879 ["rtimes"] = 8906,  
 3880 ["lthree"] = 8907,  
 3881 ["leftthreetimes"] = 8907,  
 3882 ["rthree"] = 8908,  
 3883 ["rightthreetimes"] = 8908,  
 3884 ["bsime"] = 8909,  
 3885 ["backsimeq"] = 8909,  
 3886 ["cuvee"] = 8910,  
 3887 ["curlyvee"] = 8910,  
 3888 ["cuwed"] = 8911,  
 3889 ["curlywedge"] = 8911,  
 3890 ["Sub"] = 8912,  
 3891 ["Subset"] = 8912,  
 3892 ["Sup"] = 8913,  
 3893 ["Supset"] = 8913,  
 3894 ["Cap"] = 8914,  
 3895 ["Cup"] = 8915,  
 3896 ["fork"] = 8916,  
 3897 ["pitchfork"] = 8916,  
 3898 ["epar"] = 8917,  
 3899 ["ltdot"] = 8918,  
 3900 ["lessdot"] = 8918,  
 3901 ["gtdot"] = 8919,  
 3902 ["gtrdot"] = 8919,  
 3903 ["Ll"] = 8920,  
 3904 ["Gg"] = 8921,  
 3905 ["ggg"] = 8921,  
 3906 ["leg"] = 8922,  
 3907 ["LessEqualGreater"] = 8922,  
 3908 ["lesseqgtr"] = 8922,  
 3909 ["gel"] = 8923,  
 3910 ["gtreqless"] = 8923,  
 3911 ["GreaterEqualLess"] = 8923,  
 3912 ["cuepr"] = 8926,  
 3913 ["curlyeqprec"] = 8926,  
 3914 ["cuesc"] = 8927,  
 3915 ["curlyeqsucc"] = 8927,  
 3916 ["nprcue"] = 8928,

3917 ["NotPrecedesSlantEqual"] = 8928,  
 3918 ["nsccue"] = 8929,  
 3919 ["NotSucceedsSlantEqual"] = 8929,  
 3920 ["nsqsube"] = 8930,  
 3921 ["NotSquareSubsetEqual"] = 8930,  
 3922 ["nsqsupe"] = 8931,  
 3923 ["NotSquareSupersetEqual"] = 8931,  
 3924 ["lnsim"] = 8934,  
 3925 ["gnsim"] = 8935,  
 3926 ["prnsim"] = 8936,  
 3927 ["precnsim"] = 8936,  
 3928 ["scnsim"] = 8937,  
 3929 ["succnsim"] = 8937,  
 3930 ["nltri"] = 8938,  
 3931 ["ntriangleleft"] = 8938,  
 3932 ["NotLeftTriangle"] = 8938,  
 3933 ["nrtri"] = 8939,  
 3934 ["ntriangleright"] = 8939,  
 3935 ["NotRightTriangle"] = 8939,  
 3936 ["nltrie"] = 8940,  
 3937 ["ntrianglelefteq"] = 8940,  
 3938 ["NotLeftTriangleEqual"] = 8940,  
 3939 ["nrtrie"] = 8941,  
 3940 ["ntrianglerighteq"] = 8941,  
 3941 ["NotRightTriangleEqual"] = 8941,  
 3942 ["vellip"] = 8942,  
 3943 ["ctdot"] = 8943,  
 3944 ["utdot"] = 8944,  
 3945 ["dtdot"] = 8945,  
 3946 ["disin"] = 8946,  
 3947 ["isinsv"] = 8947,  
 3948 ["isins"] = 8948,  
 3949 ["isindot"] = 8949,  
 3950 ["notinvc"] = 8950,  
 3951 ["notinvb"] = 8951,  
 3952 ["isinE"] = 8953,  
 3953 ["nisd"] = 8954,  
 3954 ["xnis"] = 8955,  
 3955 ["nis"] = 8956,  
 3956 ["notnivc"] = 8957,  
 3957 ["notnivb"] = 8958,  
 3958 ["barwed"] = 8965,  
 3959 ["barwedge"] = 8965,  
 3960 ["Barwed"] = 8966,  
 3961 ["doublebarwedge"] = 8966,  
 3962 ["lceil"] = 8968,  
 3963 ["LeftCeiling"] = 8968,

3964 ["rceil"] = 8969,  
 3965 ["RightCeiling"] = 8969,  
 3966 ["lfloor"] = 8970,  
 3967 ["LeftFloor"] = 8970,  
 3968 ["rfloor"] = 8971,  
 3969 ["RightFloor"] = 8971,  
 3970 ["drcrop"] = 8972,  
 3971 ["dlcrop"] = 8973,  
 3972 ["urcrop"] = 8974,  
 3973 ["ulcrop"] = 8975,  
 3974 ["bnot"] = 8976,  
 3975 ["proflines"] = 8978,  
 3976 ["profsurf"] = 8979,  
 3977 ["telrec"] = 8981,  
 3978 ["target"] = 8982,  
 3979 ["ulcorn"] = 8988,  
 3980 ["ulcorner"] = 8988,  
 3981 ["urcorn"] = 8989,  
 3982 ["urcorner"] = 8989,  
 3983 ["dlcorn"] = 8990,  
 3984 ["llcorner"] = 8990,  
 3985 ["drcorn"] = 8991,  
 3986 ["lrcorn"] = 8991,  
 3987 ["frown"] = 8994,  
 3988 ["sfrown"] = 8994,  
 3989 ["smile"] = 8995,  
 3990 ["ssmile"] = 8995,  
 3991 ["cylcty"] = 9005,  
 3992 ["profalar"] = 9006,  
 3993 ["topbot"] = 9014,  
 3994 ["ovbar"] = 9021,  
 3995 ["solbar"] = 9023,  
 3996 ["angzarr"] = 9084,  
 3997 ["lmoust"] = 9136,  
 3998 ["lmoustache"] = 9136,  
 3999 ["rmoust"] = 9137,  
 4000 ["rmoustache"] = 9137,  
 4001 ["tbrk"] = 9140,  
 4002 ["OverBracket"] = 9140,  
 4003 ["bbrk"] = 9141,  
 4004 ["UnderBracket"] = 9141,  
 4005 ["bbrktbrk"] = 9142,  
 4006 ["OverParenthesis"] = 9180,  
 4007 ["UnderParenthesis"] = 9181,  
 4008 ["OverBrace"] = 9182,  
 4009 ["UnderBrace"] = 9183,  
 4010 ["trpezium"] = 9186,

4011 ["elinters"] = 9191,  
4012 ["blank"] = 9251,  
4013 ["oS"] = 9416,  
4014 ["circledS"] = 9416,  
4015 ["boxh"] = 9472,  
4016 ["HorizontalLine"] = 9472,  
4017 ["boxv"] = 9474,  
4018 ["boxdr"] = 9484,  
4019 ["boxdl"] = 9488,  
4020 ["boxur"] = 9492,  
4021 ["boxul"] = 9496,  
4022 ["boxvr"] = 9500,  
4023 ["boxvl"] = 9508,  
4024 ["boxhd"] = 9516,  
4025 ["boxhu"] = 9524,  
4026 ["boxvh"] = 9532,  
4027 ["boxH"] = 9552,  
4028 ["boxV"] = 9553,  
4029 ["boxdR"] = 9554,  
4030 ["boxDr"] = 9555,  
4031 ["boxDR"] = 9556,  
4032 ["boxdL"] = 9557,  
4033 ["boxDL"] = 9558,  
4034 ["boxDL"] = 9559,  
4035 ["boxuR"] = 9560,  
4036 ["boxUr"] = 9561,  
4037 ["boxUR"] = 9562,  
4038 ["boxuL"] = 9563,  
4039 ["boxUL"] = 9564,  
4040 ["boxUL"] = 9565,  
4041 ["boxvR"] = 9566,  
4042 ["boxVr"] = 9567,  
4043 ["boxVR"] = 9568,  
4044 ["boxvL"] = 9569,  
4045 ["boxVl"] = 9570,  
4046 ["boxVL"] = 9571,  
4047 ["boxHd"] = 9572,  
4048 ["boxhD"] = 9573,  
4049 ["boxHD"] = 9574,  
4050 ["boxHu"] = 9575,  
4051 ["boxhU"] = 9576,  
4052 ["boxHU"] = 9577,  
4053 ["boxvH"] = 9578,  
4054 ["boxVh"] = 9579,  
4055 ["boxVH"] = 9580,  
4056 ["uhblk"] = 9600,  
4057 ["lhblk"] = 9604,

4058 ["block"] = 9608,  
 4059 ["blk14"] = 9617,  
 4060 ["blk12"] = 9618,  
 4061 ["blk34"] = 9619,  
 4062 ["squ"] = 9633,  
 4063 ["square"] = 9633,  
 4064 ["Square"] = 9633,  
 4065 ["squf"] = 9642,  
 4066 ["squarf"] = 9642,  
 4067 ["blacksquare"] = 9642,  
 4068 ["FilledVerySmallSquare"] = 9642,  
 4069 ["EmptyVerySmallSquare"] = 9643,  
 4070 ["rect"] = 9645,  
 4071 ["marker"] = 9646,  
 4072 ["fltns"] = 9649,  
 4073 ["xutri"] = 9651,  
 4074 ["bigtriangleup"] = 9651,  
 4075 ["utrif"] = 9652,  
 4076 ["blacktriangle"] = 9652,  
 4077 ["utri"] = 9653,  
 4078 ["triangle"] = 9653,  
 4079 ["rtrif"] = 9656,  
 4080 ["blacktriangleright"] = 9656,  
 4081 ["rtri"] = 9657,  
 4082 ["triangleright"] = 9657,  
 4083 ["xdtri"] = 9661,  
 4084 ["bigtriangledown"] = 9661,  
 4085 ["dtrif"] = 9662,  
 4086 ["blacktriangledown"] = 9662,  
 4087 ["dtri"] = 9663,  
 4088 ["triangledown"] = 9663,  
 4089 ["ltrif"] = 9666,  
 4090 ["blacktriangleleft"] = 9666,  
 4091 ["ltri"] = 9667,  
 4092 ["triangleleft"] = 9667,  
 4093 ["loz"] = 9674,  
 4094 ["lozenge"] = 9674,  
 4095 ["cir"] = 9675,  
 4096 ["tridot"] = 9708,  
 4097 ["xcirc"] = 9711,  
 4098 ["bigcirc"] = 9711,  
 4099 ["ultri"] = 9720,  
 4100 ["urtri"] = 9721,  
 4101 ["lltri"] = 9722,  
 4102 ["EmptySmallSquare"] = 9723,  
 4103 ["FilledSmallSquare"] = 9724,  
 4104 ["starf"] = 9733,

4105 ["bigstar"] = 9733,  
4106 ["star"] = 9734,  
4107 ["phone"] = 9742,  
4108 ["female"] = 9792,  
4109 ["male"] = 9794,  
4110 ["spades"] = 9824,  
4111 ["spadesuit"] = 9824,  
4112 ["clubs"] = 9827,  
4113 ["clubsuit"] = 9827,  
4114 ["hearts"] = 9829,  
4115 ["heartsuit"] = 9829,  
4116 ["diams"] = 9830,  
4117 ["diamondsuit"] = 9830,  
4118 ["sung"] = 9834,  
4119 ["flat"] = 9837,  
4120 ["natur"] = 9838,  
4121 ["natural"] = 9838,  
4122 ["sharp"] = 9839,  
4123 ["check"] = 10003,  
4124 ["checkmark"] = 10003,  
4125 ["cross"] = 10007,  
4126 ["malt"] = 10016,  
4127 ["maltese"] = 10016,  
4128 ["sext"] = 10038,  
4129 ["VerticalSeparator"] = 10072,  
4130 ["lbrkr"] = 10098,  
4131 ["rbrkr"] = 10099,  
4132 ["lobrkr"] = 10214,  
4133 ["LeftDoubleBracket"] = 10214,  
4134 ["robrkr"] = 10215,  
4135 ["RightDoubleBracket"] = 10215,  
4136 ["lang"] = 10216,  
4137 ["LeftAngleBracket"] = 10216,  
4138 ["langle"] = 10216,  
4139 ["rang"] = 10217,  
4140 ["RightAngleBracket"] = 10217,  
4141 ["rangle"] = 10217,  
4142 ["Lang"] = 10218,  
4143 ["Rang"] = 10219,  
4144 ["loang"] = 10220,  
4145 ["roang"] = 10221,  
4146 ["xlarr"] = 10229,  
4147 ["longleftarrow"] = 10229,  
4148 ["LongLeftArrow"] = 10229,  
4149 ["xrarr"] = 10230,  
4150 ["longrightarrow"] = 10230,  
4151 ["LongRightArrow"] = 10230,

4152 ["xharr"] = 10231,  
 4153 ["longleftarrow"] = 10231,  
 4154 ["LongLeftRightArrow"] = 10231,  
 4155 ["xlArr"] = 10232,  
 4156 ["Longleftarrow"] = 10232,  
 4157 ["DoubleLongLeftArrow"] = 10232,  
 4158 ["xrArr"] = 10233,  
 4159 ["Longrightarrow"] = 10233,  
 4160 ["DoubleLongRightArrow"] = 10233,  
 4161 ["xhArr"] = 10234,  
 4162 ["Longleftarrow"] = 10234,  
 4163 ["DoubleLongLeftRightArrow"] = 10234,  
 4164 ["xmap"] = 10236,  
 4165 ["longmapsto"] = 10236,  
 4166 ["dzigrarr"] = 10239,  
 4167 ["nvlArr"] = 10498,  
 4168 ["nvrArr"] = 10499,  
 4169 ["nvHarr"] = 10500,  
 4170 ["Map"] = 10501,  
 4171 ["lbarr"] = 10508,  
 4172 ["rbarr"] = 10509,  
 4173 ["bkarow"] = 10509,  
 4174 ["lBarr"] = 10510,  
 4175 ["rBarr"] = 10511,  
 4176 ["dbkarow"] = 10511,  
 4177 ["RBarr"] = 10512,  
 4178 ["drbkarow"] = 10512,  
 4179 ["DDotrahd"] = 10513,  
 4180 ["UpArrowBar"] = 10514,  
 4181 ["DownArrowBar"] = 10515,  
 4182 ["Rarrtl"] = 10518,  
 4183 ["latail"] = 10521,  
 4184 ["ratail"] = 10522,  
 4185 ["lAtail"] = 10523,  
 4186 ["rAtail"] = 10524,  
 4187 ["larrfs"] = 10525,  
 4188 ["rarrfs"] = 10526,  
 4189 ["larrbfs"] = 10527,  
 4190 ["rarrbfs"] = 10528,  
 4191 ["nwarhk"] = 10531,  
 4192 ["nearhk"] = 10532,  
 4193 ["searhk"] = 10533,  
 4194 ["hksearow"] = 10533,  
 4195 ["swarhk"] = 10534,  
 4196 ["hkswarow"] = 10534,  
 4197 ["nwnear"] = 10535,  
 4198 ["nesear"] = 10536,



4199 ["toea"] = 10536,  
4200 ["seswar"] = 10537,  
4201 ["tosa"] = 10537,  
4202 ["swnwar"] = 10538,  
4203 ["rarrc"] = 10547,  
4204 ["cudarr"] = 10549,  
4205 ["ldca"] = 10550,  
4206 ["rdca"] = 10551,  
4207 ["cudarrl"] = 10552,  
4208 ["larrpl"] = 10553,  
4209 ["curarrm"] = 10556,  
4210 ["cularrp"] = 10557,  
4211 ["rarrpl"] = 10565,  
4212 ["harrcir"] = 10568,  
4213 ["Uarrocir"] = 10569,  
4214 ["lurdshar"] = 10570,  
4215 ["ldrushar"] = 10571,  
4216 ["LeftRightVector"] = 10574,  
4217 ["RightUpDownVector"] = 10575,  
4218 ["DownLeftRightVector"] = 10576,  
4219 ["LeftUpDownVector"] = 10577,  
4220 ["LeftVectorBar"] = 10578,  
4221 ["RightVectorBar"] = 10579,  
4222 ["RightUpVectorBar"] = 10580,  
4223 ["RightDownVectorBar"] = 10581,  
4224 ["DownLeftVectorBar"] = 10582,  
4225 ["DownRightVectorBar"] = 10583,  
4226 ["LeftUpVectorBar"] = 10584,  
4227 ["LeftDownVectorBar"] = 10585,  
4228 ["LeftTeeVector"] = 10586,  
4229 ["RightTeeVector"] = 10587,  
4230 ["RightUpTeeVector"] = 10588,  
4231 ["RightDownTeeVector"] = 10589,  
4232 ["DownLeftTeeVector"] = 10590,  
4233 ["DownRightTeeVector"] = 10591,  
4234 ["LeftUpTeeVector"] = 10592,  
4235 ["LeftDownTeeVector"] = 10593,  
4236 ["lHar"] = 10594,  
4237 ["uHar"] = 10595,  
4238 ["rHar"] = 10596,  
4239 ["dHar"] = 10597,  
4240 ["luruhar"] = 10598,  
4241 ["ldrdhar"] = 10599,  
4242 ["ruluhar"] = 10600,  
4243 ["rdldhar"] = 10601,  
4244 ["lharul"] = 10602,  
4245 ["llhard"] = 10603,

4246 ["rharul"] = 10604,  
4247 ["lrhard"] = 10605,  
4248 ["udhar"] = 10606,  
4249 ["UpEquilibrium"] = 10606,  
4250 ["duhar"] = 10607,  
4251 ["ReverseUpEquilibrium"] = 10607,  
4252 ["RoundImplies"] = 10608,  
4253 ["erarr"] = 10609,  
4254 ["simrarr"] = 10610,  
4255 ["larrsim"] = 10611,  
4256 ["rarrsim"] = 10612,  
4257 ["rarrap"] = 10613,  
4258 ["ltlarr"] = 10614,  
4259 ["gtrarr"] = 10616,  
4260 ["subrarr"] = 10617,  
4261 ["suplarr"] = 10619,  
4262 ["lfisht"] = 10620,  
4263 ["rfisht"] = 10621,  
4264 ["ufisht"] = 10622,  
4265 ["dfisht"] = 10623,  
4266 ["lopar"] = 10629,  
4267 ["ropar"] = 10630,  
4268 ["lbrke"] = 10635,  
4269 ["rbrke"] = 10636,  
4270 ["lbrkslu"] = 10637,  
4271 ["rbrksld"] = 10638,  
4272 ["lbrksld"] = 10639,  
4273 ["rbrkslu"] = 10640,  
4274 ["langd"] = 10641,  
4275 ["rangd"] = 10642,  
4276 ["lparlt"] = 10643,  
4277 ["rpargt"] = 10644,  
4278 ["gtlPar"] = 10645,  
4279 ["ltrPar"] = 10646,  
4280 ["vzigzag"] = 10650,  
4281 ["vangrt"] = 10652,  
4282 ["angrtvbd"] = 10653,  
4283 ["ange"] = 10660,  
4284 ["range"] = 10661,  
4285 ["dwangle"] = 10662,  
4286 ["uwangle"] = 10663,  
4287 ["angmsdaa"] = 10664,  
4288 ["angmsdab"] = 10665,  
4289 ["angmsdac"] = 10666,  
4290 ["angmsdad"] = 10667,  
4291 ["angmsdae"] = 10668,  
4292 ["angmsdaf"] = 10669,

4293 ["angmsdag"] = 10670,  
4294 ["angmsdah"] = 10671,  
4295 ["bemptyv"] = 10672,  
4296 ["demptyv"] = 10673,  
4297 ["cemptyv"] = 10674,  
4298 ["raemptyv"] = 10675,  
4299 ["laemptyv"] = 10676,  
4300 ["ohbar"] = 10677,  
4301 ["omid"] = 10678,  
4302 ["opar"] = 10679,  
4303 ["operp"] = 10681,  
4304 ["olcross"] = 10683,  
4305 ["odsold"] = 10684,  
4306 ["olcir"] = 10686,  
4307 ["ofcir"] = 10687,  
4308 ["olt"] = 10688,  
4309 ["ogt"] = 10689,  
4310 ["cirscir"] = 10690,  
4311 ["cirE"] = 10691,  
4312 ["solb"] = 10692,  
4313 ["bsolb"] = 10693,  
4314 ["boxbox"] = 10697,  
4315 ["trish"] = 10701,  
4316 ["rtriltri"] = 10702,  
4317 ["LeftTriangleBar"] = 10703,  
4318 ["RightTriangleBar"] = 10704,  
4319 ["race"] = 10714,  
4320 ["iinfin"] = 10716,  
4321 ["infintie"] = 10717,  
4322 ["nvinfin"] = 10718,  
4323 ["eparsl"] = 10723,  
4324 ["smeparsl"] = 10724,  
4325 ["eqvparsl"] = 10725,  
4326 ["lozf"] = 10731,  
4327 ["blacklozenge"] = 10731,  
4328 ["RuleDelayed"] = 10740,  
4329 ["dsol"] = 10742,  
4330 ["xodot"] = 10752,  
4331 ["bigodot"] = 10752,  
4332 ["xoplus"] = 10753,  
4333 ["bigoplus"] = 10753,  
4334 ["xotime"] = 10754,  
4335 ["bigotimes"] = 10754,  
4336 ["xuplus"] = 10756,  
4337 ["biguplus"] = 10756,  
4338 ["xsqcup"] = 10758,  
4339 ["bigsqcup"] = 10758,

4340 ["qint"] = 10764,  
4341 ["iiiint"] = 10764,  
4342 ["fpartint"] = 10765,  
4343 ["cirfnint"] = 10768,  
4344 ["awint"] = 10769,  
4345 ["rppolint"] = 10770,  
4346 ["scpolint"] = 10771,  
4347 ["npolint"] = 10772,  
4348 ["pointint"] = 10773,  
4349 ["quatint"] = 10774,  
4350 ["intlarhk"] = 10775,  
4351 ["pluscir"] = 10786,  
4352 ["plusacir"] = 10787,  
4353 ["simplus"] = 10788,  
4354 ["plusdu"] = 10789,  
4355 ["plussim"] = 10790,  
4356 ["plustwo"] = 10791,  
4357 ["mcomma"] = 10793,  
4358 ["minusdu"] = 10794,  
4359 ["loplus"] = 10797,  
4360 ["roplus"] = 10798,  
4361 ["Cross"] = 10799,  
4362 ["timesd"] = 10800,  
4363 ["timesbar"] = 10801,  
4364 ["smashp"] = 10803,  
4365 ["lotimes"] = 10804,  
4366 ["rotimes"] = 10805,  
4367 ["otimesas"] = 10806,  
4368 ["Otimes"] = 10807,  
4369 ["odiv"] = 10808,  
4370 ["triplus"] = 10809,  
4371 ["triminus"] = 10810,  
4372 ["tritime"] = 10811,  
4373 ["iproduct"] = 10812,  
4374 ["intprod"] = 10812,  
4375 ["amalg"] = 10815,  
4376 ["capdot"] = 10816,  
4377 ["ncup"] = 10818,  
4378 ["ncap"] = 10819,  
4379 ["capand"] = 10820,  
4380 ["cupor"] = 10821,  
4381 ["cupcap"] = 10822,  
4382 ["capcup"] = 10823,  
4383 ["cupbrcap"] = 10824,  
4384 ["capbrcup"] = 10825,  
4385 ["cupcup"] = 10826,  
4386 ["capcap"] = 10827,

4387 ["ccups"] = 10828,  
 4388 ["ccaps"] = 10829,  
 4389 ["ccupssm"] = 10832,  
 4390 ["And"] = 10835,  
 4391 ["Or"] = 10836,  
 4392 ["andand"] = 10837,  
 4393 ["oror"] = 10838,  
 4394 ["orslope"] = 10839,  
 4395 ["andslope"] = 10840,  
 4396 ["andv"] = 10842,  
 4397 ["orv"] = 10843,  
 4398 ["andd"] = 10844,  
 4399 ["ord"] = 10845,  
 4400 ["wedbar"] = 10847,  
 4401 ["sdote"] = 10854,  
 4402 ["simdot"] = 10858,  
 4403 ["congdot"] = 10861,  
 4404 ["easter"] = 10862,  
 4405 ["apacir"] = 10863,  
 4406 ["apE"] = 10864,  
 4407 ["eplus"] = 10865,  
 4408 ["pluse"] = 10866,  
 4409 ["Esim"] = 10867,  
 4410 ["Colone"] = 10868,  
 4411 ["Equal"] = 10869,  
 4412 ["eDDot"] = 10871,  
 4413 ["ddotseq"] = 10871,  
 4414 ["equivDD"] = 10872,  
 4415 ["ltcir"] = 10873,  
 4416 ["gtcir"] = 10874,  
 4417 ["ltquest"] = 10875,  
 4418 ["gtquest"] = 10876,  
 4419 ["les"] = 10877,  
 4420 ["LessSlantEqual"] = 10877,  
 4421 ["leqslant"] = 10877,  
 4422 ["ges"] = 10878,  
 4423 ["GreaterSlantEqual"] = 10878,  
 4424 ["geqslant"] = 10878,  
 4425 ["lesdot"] = 10879,  
 4426 ["gesdot"] = 10880,  
 4427 ["lesdoto"] = 10881,  
 4428 ["gesdoto"] = 10882,  
 4429 ["lesdotor"] = 10883,  
 4430 ["gesdotol"] = 10884,  
 4431 ["lap"] = 10885,  
 4432 ["lessapprox"] = 10885,  
 4433 ["gap"] = 10886,

4434 ["gtrapprox"] = 10886,  
4435 ["lne"] = 10887,  
4436 ["lneq"] = 10887,  
4437 ["gne"] = 10888,  
4438 ["gneq"] = 10888,  
4439 ["lnap"] = 10889,  
4440 ["lnapprox"] = 10889,  
4441 ["gnap"] = 10890,  
4442 ["gnapprox"] = 10890,  
4443 ["lEg"] = 10891,  
4444 ["lesseqqtr"] = 10891,  
4445 ["gEl"] = 10892,  
4446 ["gtreqqless"] = 10892,  
4447 ["lsime"] = 10893,  
4448 ["gsime"] = 10894,  
4449 ["lsimg"] = 10895,  
4450 ["gsiml"] = 10896,  
4451 ["lgE"] = 10897,  
4452 ["glE"] = 10898,  
4453 ["lesges"] = 10899,  
4454 ["gesles"] = 10900,  
4455 ["els"] = 10901,  
4456 ["eqslantless"] = 10901,  
4457 ["egs"] = 10902,  
4458 ["eqslantgtr"] = 10902,  
4459 ["elsdot"] = 10903,  
4460 ["egsdot"] = 10904,  
4461 ["el"] = 10905,  
4462 ["eg"] = 10906,  
4463 ["siml"] = 10909,  
4464 ["simg"] = 10910,  
4465 ["simLE"] = 10911,  
4466 ["simGE"] = 10912,  
4467 ["LessLess"] = 10913,  
4468 ["GreaterGreater"] = 10914,  
4469 ["glj"] = 10916,  
4470 ["gla"] = 10917,  
4471 ["ltcc"] = 10918,  
4472 ["gtcc"] = 10919,  
4473 ["lescc"] = 10920,  
4474 ["gescc"] = 10921,  
4475 ["smt"] = 10922,  
4476 ["lat"] = 10923,  
4477 ["smtte"] = 10924,  
4478 ["late"] = 10925,  
4479 ["bumpE"] = 10926,  
4480 ["pre"] = 10927,

4481 ["preceq"] = 10927,  
4482 ["PrecedesEqual"] = 10927,  
4483 ["sce"] = 10928,  
4484 ["succeq"] = 10928,  
4485 ["SucceedsEqual"] = 10928,  
4486 ["prE"] = 10931,  
4487 ["scE"] = 10932,  
4488 ["prnE"] = 10933,  
4489 ["precneqq"] = 10933,  
4490 ["scnE"] = 10934,  
4491 ["succneqq"] = 10934,  
4492 ["prap"] = 10935,  
4493 ["precapprox"] = 10935,  
4494 ["scap"] = 10936,  
4495 ["succapprox"] = 10936,  
4496 ["prnap"] = 10937,  
4497 ["precnapprox"] = 10937,  
4498 ["scnap"] = 10938,  
4499 ["succnapprox"] = 10938,  
4500 ["Pr"] = 10939,  
4501 ["Sc"] = 10940,  
4502 ["subdot"] = 10941,  
4503 ["supdot"] = 10942,  
4504 ["subplus"] = 10943,  
4505 ["supplus"] = 10944,  
4506 ["submult"] = 10945,  
4507 ["supmult"] = 10946,  
4508 ["subedot"] = 10947,  
4509 ["supedot"] = 10948,  
4510 ["subE"] = 10949,  
4511 ["subseteqq"] = 10949,  
4512 ["supE"] = 10950,  
4513 ["supseteqq"] = 10950,  
4514 ["subsim"] = 10951,  
4515 ["supsim"] = 10952,  
4516 ["subnE"] = 10955,  
4517 ["subsetneqq"] = 10955,  
4518 ["supnE"] = 10956,  
4519 ["supsetneqq"] = 10956,  
4520 ["csub"] = 10959,  
4521 ["csup"] = 10960,  
4522 ["csube"] = 10961,  
4523 ["csupe"] = 10962,  
4524 ["subsup"] = 10963,  
4525 ["supsub"] = 10964,  
4526 ["subsub"] = 10965,  
4527 ["supsup"] = 10966,

4528 ["suphsub"] = 10967,  
4529 ["supdsub"] = 10968,  
4530 ["forkv"] = 10969,  
4531 ["topfork"] = 10970,  
4532 ["mlcp"] = 10971,  
4533 ["Dashv"] = 10980,  
4534 ["DoubleLeftTee"] = 10980,  
4535 ["Vdashl"] = 10982,  
4536 ["Barv"] = 10983,  
4537 ["vBar"] = 10984,  
4538 ["vBarv"] = 10985,  
4539 ["Vbar"] = 10987,  
4540 ["Not"] = 10988,  
4541 ["bNot"] = 10989,  
4542 ["rmid"] = 10990,  
4543 ["cirmid"] = 10991,  
4544 ["midcir"] = 10992,  
4545 ["topcir"] = 10993,  
4546 ["nhpar"] = 10994,  
4547 ["parsim"] = 10995,  
4548 ["parsl"] = 11005,  
4549 ["fflig"] = 64256,  
4550 ["filig"] = 64257,  
4551 ["fllig"] = 64258,  
4552 ["ffilig"] = 64259,  
4553 ["ffllig"] = 64260,  
4554 ["Ascr"] = 119964,  
4555 ["Cscr"] = 119966,  
4556 ["Dscr"] = 119967,  
4557 ["Gscr"] = 119970,  
4558 ["Jscr"] = 119973,  
4559 ["Kscr"] = 119974,  
4560 ["Nscr"] = 119977,  
4561 ["Oscr"] = 119978,  
4562 ["Pscr"] = 119979,  
4563 ["Qscr"] = 119980,  
4564 ["Sscr"] = 119982,  
4565 ["Tscr"] = 119983,  
4566 ["Uscr"] = 119984,  
4567 ["Vscr"] = 119985,  
4568 ["Wscr"] = 119986,  
4569 ["Xscr"] = 119987,  
4570 ["Yscr"] = 119988,  
4571 ["Zscr"] = 119989,  
4572 ["ascr"] = 119990,  
4573 ["bscr"] = 119991,  
4574 ["cscr"] = 119992,



4575 ["dscr"] = 119993,  
4576 ["fscr"] = 119995,  
4577 ["hscr"] = 119997,  
4578 ["iscr"] = 119998,  
4579 ["jscr"] = 119999,  
4580 ["kscr"] = 120000,  
4581 ["lscr"] = 120001,  
4582 ["mscr"] = 120002,  
4583 ["nscr"] = 120003,  
4584 ["pscr"] = 120005,  
4585 ["qscr"] = 120006,  
4586 ["rscr"] = 120007,  
4587 ["sscr"] = 120008,  
4588 ["tscr"] = 120009,  
4589 ["uscr"] = 120010,  
4590 ["vscr"] = 120011,  
4591 ["wscr"] = 120012,  
4592 ["xscr"] = 120013,  
4593 ["yscr"] = 120014,  
4594 ["zscr"] = 120015,  
4595 ["Afr"] = 120068,  
4596 ["Bfr"] = 120069,  
4597 ["Dfr"] = 120071,  
4598 ["Efr"] = 120072,  
4599 ["Ffr"] = 120073,  
4600 ["Gfr"] = 120074,  
4601 ["Jfr"] = 120077,  
4602 ["Kfr"] = 120078,  
4603 ["Lfr"] = 120079,  
4604 ["Mfr"] = 120080,  
4605 ["Nfr"] = 120081,  
4606 ["Ofr"] = 120082,  
4607 ["Pfr"] = 120083,  
4608 ["Qfr"] = 120084,  
4609 ["Sfr"] = 120086,  
4610 ["Tfr"] = 120087,  
4611 ["Ufr"] = 120088,  
4612 ["Vfr"] = 120089,  
4613 ["Wfr"] = 120090,  
4614 ["Xfr"] = 120091,  
4615 ["Yfr"] = 120092,  
4616 ["afr"] = 120094,  
4617 ["bfr"] = 120095,  
4618 ["cfr"] = 120096,  
4619 ["dfr"] = 120097,  
4620 ["efr"] = 120098,  
4621 ["ffr"] = 120099,

4622 ["gfr"] = 120100,  
4623 ["hfr"] = 120101,  
4624 ["ifr"] = 120102,  
4625 ["jfr"] = 120103,  
4626 ["kfr"] = 120104,  
4627 ["lfr"] = 120105,  
4628 ["mfr"] = 120106,  
4629 ["nfr"] = 120107,  
4630 ["ofr"] = 120108,  
4631 ["pfr"] = 120109,  
4632 ["qfr"] = 120110,  
4633 ["rfr"] = 120111,  
4634 ["sfr"] = 120112,  
4635 ["tfr"] = 120113,  
4636 ["ufr"] = 120114,  
4637 ["vfr"] = 120115,  
4638 ["wfr"] = 120116,  
4639 ["xfr"] = 120117,  
4640 ["yfr"] = 120118,  
4641 ["zfr"] = 120119,  
4642 ["Aopf"] = 120120,  
4643 ["Bopf"] = 120121,  
4644 ["Dopf"] = 120123,  
4645 ["Eopf"] = 120124,  
4646 ["Fopf"] = 120125,  
4647 ["Gopf"] = 120126,  
4648 ["Iopf"] = 120128,  
4649 ["Jopf"] = 120129,  
4650 ["Kopf"] = 120130,  
4651 ["Lopf"] = 120131,  
4652 ["Mopf"] = 120132,  
4653 ["Oopf"] = 120134,  
4654 ["Sopf"] = 120138,  
4655 ["Topf"] = 120139,  
4656 ["Uopf"] = 120140,  
4657 ["Vopf"] = 120141,  
4658 ["Wopf"] = 120142,  
4659 ["Xopf"] = 120143,  
4660 ["Yopf"] = 120144,  
4661 ["aopf"] = 120146,  
4662 ["bopf"] = 120147,  
4663 ["copf"] = 120148,  
4664 ["dopf"] = 120149,  
4665 ["eopf"] = 120150,  
4666 ["fopf"] = 120151,  
4667 ["gopf"] = 120152,  
4668 ["hopf"] = 120153,

```

4669 ["iopf"] = 120154,
4670 ["jopf"] = 120155,
4671 ["kopf"] = 120156,
4672 ["lopf"] = 120157,
4673 ["mopf"] = 120158,
4674 ["nopf"] = 120159,
4675 ["oopf"] = 120160,
4676 ["popf"] = 120161,
4677 ["qopf"] = 120162,
4678 ["ropf"] = 120163,
4679 ["sopf"] = 120164,
4680 ["topf"] = 120165,
4681 ["uopf"] = 120166,
4682 ["vopf"] = 120167,
4683 ["wopf"] = 120168,
4684 ["xopf"] = 120169,
4685 ["yopf"] = 120170,
4686 ["zopf"] = 120171,
4687 }

```

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

```

4688 function entities.dec_entity(s)
4689 return unicode.utf8.char(tonumber(s))
4690 end

```

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

```

4691 function entities.hex_entity(s)
4692 return unicode.utf8.char(tonumber("0x"..s))
4693 end

```

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

```

4694 function entities.char_entity(s)
4695 local n = character_entities[s]
4696 if n == nil then
4697 return "&" .. s .. ";"
4698 end
4699 return unicode.utf8.char(n)
4700 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 Lunamark Lua module.

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

```
4701 M.writer = {}
```

The `writer.new` method creates and returns a new TeX 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  $\langle member \rangle$ s as `writer->\langle member \rangle`. All member variables are immutable unless explicitly stated otherwise.

```
4702 function M.writer.new(options)
4703 local self = {}
```

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

```
4704 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.

```
4705 local slice_specifiers = {}
4706 for specifier in options.slice:gmatch("[^%s]+") do
4707 table.insert(slice_specifiers, specifier)
4708 end
4709
4710 if #slice_specifiers == 2 then
4711 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4712 local slice_begin_type = self.slice_begin:sub(1, 1)
4713 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4714 self.slice_begin = "^" .. self.slice_begin
4715 end
4716 local slice_end_type = self.slice_end:sub(1, 1)
4717 if slice_end_type ~= "^" and slice_end_type ~= "$" then
4718 self.slice_end = "$" .. self.slice_end
4719 end
4720 elseif #slice_specifiers == 1 then
4721 self.slice_begin = "^" .. slice_specifiers[1]
4722 self.slice_end = "$" .. slice_specifiers[1]
4723 end
4724
4725 if self.slice_begin == "^" and self.slice_end ~= "^" then
4726 self.is_writing = true
```

```

4727 else
4728 self.is_writing = false
4729 end

Define writer->suffix as the suffix of the produced cache files.
4730 self.suffix = ".tex"

Define writer->space as the output format of a space character.
4731 self.space = " "

Define writer->nbsp as the output format of a non-breaking space character.
4732 self.nbsp = "\\markdownRendererNbsp{}"

Define writer->plain as a function that will transform an input plain text block
s to the output format.
4733 function self.plain(s)
4734 return s
4735 end

Define writer->paragraph as a function that will transform an input paragraph
s to the output format.
4736 function self.paragraph(s)
4737 if not self.is_writing then return "" end
4738 return s
4739 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.
4740 function self.pack(name)
4741 return "[[input]] .. name .. [[relax]]"
4742 end

Define writer->interblocksep as the output format of a block element separator.
4743 function self.interblocksep()
4744 if not self.is_writing then return "" end
4745 return "\\markdownRendererInterblockSeparator\n{}"
4746 end

Define writer->linebreak as the output format of a forced line break.
4747 self.linebreak = "\\markdownRendererLineBreak\n{}"

Define writer->ellipsis as the output format of an ellipsis.
4748 self.ellipsis = "\\markdownRendererEllipsis{}"

Define writer->thematic_break as the output format of a thematic break.
4749 function self.thematic_break()
4750 if not self.is_writing then return "" end
4751 return "\\markdownRendererThematicBreak{}"
4752 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.

```

4753 self.escaped_uri_chars = {
4754 [{""] = "\\markdownRendererLeftBrace{}",
4755 ["}"] = "\\markdownRendererRightBrace{}",
4756 [%"] = "\\markdownRendererPercentSign{}",
4757 ["\\""] = "\\markdownRendererBackslash{}",
4758 }
4759 self.escaped_minimal_strings = {
4760 ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4761 ["☒"] = "\\markdownRendererTickedBox{}",
4762 ["☐"] = "\\markdownRendererHalfTickedBox{}",
4763 ["□"] = "\\markdownRendererUntickedBox{}",
4764 }

```

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

```

4765 self.escaped_chars = {
4766 [{""] = "\\markdownRendererLeftBrace{}",
4767 ["}"] = "\\markdownRendererRightBrace{}",
4768 [%"] = "\\markdownRendererPercentSign{}",
4769 ["\\""] = "\\markdownRendererBackslash{}",
4770 [#"] = "\\markdownRendererHash{}",
4771 [$"] = "\\markdownRendererDollarSign{}",
4772 [&"] = "\\markdownRendererAmpersand{}",
4773 [_"] = "\\markdownRendererUnderscore{}",
4774 ["^"] = "\\markdownRendererCircumflex{}",
4775 ["~"] = "\\markdownRendererTilde{}",
4776 [|"] = "\\markdownRendererPipe{}",
4777 }

```

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

```

4778 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4779 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4780 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.

```

4781 if options.hybrid then
4782 self.string = self.escape_minimal

```

```

4783 self.uri = self.escape_minimal
4784 else
4785 self.string = self.escape
4786 self.uri = self.escape_uri
4787 end

```

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

```

4788 function self.code(s)
4789 return {"\\markdownRendererCodeSpan{" ,self.escape(s),"}"}
4790 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.

```

4791 function self.link(lab,src,tit)
4792 return {"\\markdownRendererLink{" ,lab,"} ",
4793 "{" ,self.escape(src),"} ",
4794 "{" ,self.uri(src),"} ",
4795 "{" ,self.string(tit or ""),""}
4796 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.

```

4797 function self.image(lab,src,tit)
4798 return {"\\markdownRendererImage{" ,lab,"} ",
4799 "{" ,self.string(src),"} ",
4800 "{" ,self.uri(src),"} ",
4801 "{" ,self.string(tit or ""),""}
4802 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.

```

4803 function self.bulletlist(items,tight)
4804 if not self.is_writing then return "" end
4805 local buffer = {}
4806 for _,item in ipairs(items) do
4807 buffer[#buffer + 1] = self.bulletitem(item)
4808 end
4809 local contents = util.intersperse(buffer,"\n")
4810 if tight and options.tightLists then
4811 return {"\\markdownRendererUlBeginTight\n",contents,
4812 "\n\\markdownRendererUlEndTight "}
4813 else
4814 return {"\\markdownRendererUlBegin\n",contents,
4815 "\n\\markdownRendererUlEnd "}

```

```
4816 end
4817 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.

```
4818 function self.bulletitem(s)
4819 return {"\\markdownRendererULItem ",s,
4820 "\\markdownRendererULItemEnd "}
4821 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.

```
4822 function self.orderedlist(items,tight,startnum)
4823 if not self.is_writing then return "" end
4824 local buffer = {}
4825 local num = startnum
4826 for _,item in ipairs(items) do
4827 buffer[#buffer + 1] = self.ordereditem(item,num)
4828 if num ~= nil then
4829 num = num + 1
4830 end
4831 end
4832 local contents = util.intersperse(buffer,"\n")
4833 if tight and options.tightLists then
4834 return {"\\markdownRendererOLBeginTight\n",contents,
4835 "\n\\markdownRendererOLEndTight "}
4836 else
4837 return {"\\markdownRendererOLBegin\n",contents,
4838 "\n\\markdownRendererOLEnd "}
4839 end
4840 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.

```
4841 function self.ordereditem(s,num)
4842 if num ~= nil then
4843 return {"\\markdownRendererOLItemWithNumber{" ,num,"} ",s,
4844 "\\markdownRendererOLItemEnd "}
4845 else
4846 return {"\\markdownRendererOLItem ",s,
4847 "\\markdownRendererOLItemEnd "}
4848 end
4849 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.

```
4850 function self.inline_html_comment(contents)
4851 return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
4852 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.

```
4853 function self.block_html_comment(contents)
4854 if not self.is_writing then return "" end
4855 return {"\\markdownRendererBlockHtmlCommentBegin\\n",contents,
4856 "\\n\\markdownRendererBlockHtmlCommentEnd "}
4857 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.

```
4858 function self.inline_html_tag(contents)
4859 return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
4860 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.

```
4861 function self.block_html_element(s)
4862 if not self.is_writing then return "" end
4863 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4864 return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
4865 end
```

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

```
4866 function self.emphasis(s)
4867 return {"\\markdownRendererEmphasis{" ,s,"}"}
4868 end
```

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

```
4869 function self.checkbox(f)
4870 if f == 1.0 then
4871 return "☒ "
4872 elseif f == 0.0 then
4873 return "☐ "
4874 else
4875 return "☐ "
4876 end
```

```
4877 end
```

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

```
4878 function self.strong(s)
4879 return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4880 end
```

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

```
4881 function self.blockquote(s)
4882 if #util.ropetostring(s) == 0 then return "" end
4883 return {"\\markdownRendererBlockQuoteBegin\n",s,
4884 "\n\\markdownRendererBlockQuoteEnd "}
4885 end
```

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

```
4886 function self.verbatim(s)
4887 if not self.is_writing then return "" end
4888 local name = util.cache_verbatim(options.cacheDir, s)
4889 return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4890 end
```

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

```
4891 function self.document(d)
4892 local active_attributes = self.active_attributes
4893 local buf = {"\\markdownRendererDocumentBegin\n", d}
4894
4895 -- pop attributes for sections that have ended
4896 if options.headerAttributes and self.is_writing then
4897 while #active_attributes > 0 do
4898 local attributes = active_attributes[#active_attributes]
4899 if #attributes > 0 then
4900 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4901 end
4902 table.remove(active_attributes, #active_attributes)
4903 end
4904 end
4905
4906 table.insert(buf, "\\markdownRendererDocumentEnd")
4907
4908 return buf
4909 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.

```
4910 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.

```
4911 function self.heading(s, level, attributes)
4912 attributes = attributes or {}
4913 for i = 1, #attributes do
4914 attributes[attributes[i]] = true
4915 end
4916
4917 local active_attributes = self.active_attributes
4918 local slice_begin_type = self.slice_begin:sub(1, 1)
4919 local slice_begin_identifier = self.slice_begin:sub(2) or ""
4920 local slice_end_type = self.slice_end:sub(1, 1)
4921 local slice_end_identifier = self.slice_end:sub(2) or ""
4922
4923 local buf = {}
4924
4925 -- push empty attributes for implied sections
4926 while #active_attributes < level-1 do
4927 table.insert(active_attributes, {})
4928 end
4929
4930 -- pop attributes for sections that have ended
4931 while #active_attributes >= level do
4932 local active_identifiers = active_attributes[#active_attributes]
4933 -- tear down all active attributes at slice end
4934 if active_identifiers["#" .. slice_end_identifier] ~= nil
4935 and slice_end_type == "$" then
4936 for header_level = #active_attributes, 1, -1 do
4937 if options.headerAttributes and #active_attributes[header_level] > 0 then
4938 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4939 end
4940 end
4941 self.is_writing = false
4942 end
4943 table.remove(active_attributes, #active_attributes)
4944 if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4945 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4946 end
4947 -- apply all active attributes at slice beginning
4948 if active_identifiers["#" .. slice_begin_identifier] ~= nil
4949 and slice_begin_type == "$" then
4950 for header_level = 1, #active_attributes do
4951 if options.headerAttributes and #active_attributes[header_level] > 0 then
4952 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4953 end
4954 end
4955 self.is_writing = true

```

```

4956 end
4957 end
4958
4959 -- tear down all active attributes at slice end
4960 if attributes["#" .. slice_end_identifiler] ~= nil
4961 and slice_end_type == "^" then
4962 for header_level = #active_attributes, 1, -1 do
4963 if options.headerAttributes and #active_attributes[header_level] > 0 then
4964 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4965 end
4966 end
4967 self.is_writing = false
4968 end
4969
4970 -- push attributes for the new section
4971 table.insert(active_attributes, attributes)
4972 if self.is_writing and options.headerAttributes and #attributes > 0 then
4973 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4974 end
4975
4976 -- apply all active attributes at slice beginning
4977 if attributes["#" .. slice_begin_identifiler] ~= nil
4978 and slice_begin_type == "^" then
4979 for header_level = 1, #active_attributes do
4980 if options.headerAttributes and #active_attributes[header_level] > 0 then
4981 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4982 end
4983 end
4984 self.is_writing = true
4985 end
4986
4987 if self.is_writing then
4988 table.sort(attributes)
4989 local key, value
4990 for i = 1, #attributes do
4991 if attributes[i]:sub(1, 1) == "#" then
4992 table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
4993 attributes[i]:sub(2), "}")})
4994 elseif attributes[i]:sub(1, 1) == "." then
4995 table.insert(buf, {"\\markdownRendererAttributeClassName{" ,
4996 attributes[i]:sub(2), "}")})
4997 else
4998 key, value = attributes[i]:match("([^\s=]+)%s*=%s*(.*)")
4999 table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5000 key, "{", value, "}")})
5001 end
5002 end

```

```

5003 end
5004
5005 local cmd
5006 level = level + options.shiftHeadings
5007 if level <= 1 then
5008 cmd = "\\markdownRendererHeadingOne"
5009 elseif level == 2 then
5010 cmd = "\\markdownRendererHeadingTwo"
5011 elseif level == 3 then
5012 cmd = "\\markdownRendererHeadingThree"
5013 elseif level == 4 then
5014 cmd = "\\markdownRendererHeadingFour"
5015 elseif level == 5 then
5016 cmd = "\\markdownRendererHeadingFive"
5017 elseif level >= 6 then
5018 cmd = "\\markdownRendererHeadingSix"
5019 else
5020 cmd = ""
5021 end
5022 if self.is_writing then
5023 table.insert(buf, {cmd, "{", s, "}"})
5024 end
5025
5026 return buf
5027 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.

```

5028 function self.get_state()
5029 return {
5030 is_writing=self.is_writing,
5031 active_attributes={table.unpack(self.active_attributes)},
5032 }
5033 end

```

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

```

5034 function self.set_state(s)
5035 local previous_state = self.get_state()
5036 for key, value in pairs(s) do
5037 self[key] = value
5038 end
5039 return previous_state
5040 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`.

```

5041 function self.defer_call(f)
5042 local previous_state = self.get_state()
5043 return function(...)
5044 local state = self.set_state(previous_state)
5045 local return_value = f(...)
5046 self.set_state(state)
5047 return return_value
5048 end
5049 end
5050
5051 return self
5052 end

```

### 3.1.4 Parsers

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

```

5053 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

5054 parsers.percent = P("%")
5055 parsers.at = P("@")
5056 parsers.comma = P(",")
5057 parsers.asterisk = P("*")
5058 parsers.dash = P("-")
5059 parsers.plus = P("+")
5060 parsers.underscore = P("_")
5061 parsers.period = P(".")
5062 parsers.hash = P("#")
5063 parsers.ampersand = P("&")
5064 parsers.backtick = P("`")
5065 parsers.less = P("<")
5066 parsers.more = P(">")
5067 parsers.space = P(" ")
5068 parsers.squote = P("'")
5069 parsers.dquote = P('"')
5070 parsers.lparent = P("(")
5071 parsers.rparent = P(")")
5072 parsers.lbracket = P("[")
5073 parsers.rbracket = P("]")
5074 parsers.lbrace = P("{")
5075 parsers.rbrace = P("}")
5076 parsers.circumflex = P("^")
5077 parsers.slash = P("/")
5078 parsers.equal = P("=")
5079 parsers.colon = P(":")

```

```

5080 parsers.semicolon = P(";")
5081 parsers.exclamation = P("!")
5082 parsers.pipe = P("|")
5083 parsers.tilde = P("~")
5084 parsers.backslash = P("\\")
5085 parsers.tab = P("\t")
5086 parsers.newline = P("\n")
5087 parsers.tightblocksep = P("\001")
5088
5089 parsers.digit = R("09")
5090 parsers.hexdigit = R("09","af","AF")
5091 parsers.letter = R("AZ","az")
5092 parsers.alphanumeric = R("AZ","az","09")
5093 parsers.keyword = parsers.letter
5094 * parsers.alphanumeric^0
5095 parsers.internal_punctuation = S(";,.,?")
5096
5097 parsers.doubleasterisks = P("**")
5098 parsers.doubleunderscores = P("__")
5099 parsers.doubletildes = P("~~")
5100 parsers.fourspaces = P(" ")
5101
5102 parsers.any = P(1)
5103 parsers.fail = parsers.any - 1
5104
5105 parsers.escapable = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
5106 parsers.anyescaped = parsers.backslash / " " * parsers.escapable
5107 + parsers.any
5108
5109 parsers.spacechar = S("\t ")
5110 parsers.spacing = S(" \n\r\t")
5111 parsers.nonspacechar = parsers.any - parsers.spacing
5112 parsers.optionalspace = parsers.spacechar^0
5113
5114 parsers.normalchar = parsers.any - (V("SpecialChar")
5115 + parsers.spacing
5116 + parsers.tightblocksep)
5117 parsers.eof = -parsers.any
5118 parsers.nonindentpace = parsers.space^-3 * - parsers.spacechar
5119 parsers.indent = parsers.space^-3 * parsers.tab
5120 + parsers.fourspaces / " "
5121 parsers.linechar = P(1 - parsers.newline)
5122
5123 parsers.blankline = parsers.optionalspace
5124 * parsers.newline / "\n"
5125 parsers.blanklines = parsers.blankline^0
5126 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0

```

```

5127 parsers.indentedline = parsers.indent /""
5128 * C(parsers.linechar^1 * parsers.newline^-
1)
5129 parsers.optionallyindentedline = parsers.indent^-1 /""
5130 * C(parsers.linechar^1 * parsers.newline^-
1)
5131 parsers.sp = parsers.spacing^0
5132 parsers.spnl = parsers.optionalspace
5133 * (parsers.newline * parsers.optionalspace)^-
1
5134 parsers.line = parsers.linechar^0 * parsers.newline
5135 parsers.nonemptyline = parsers.line - parsers.blankline

```

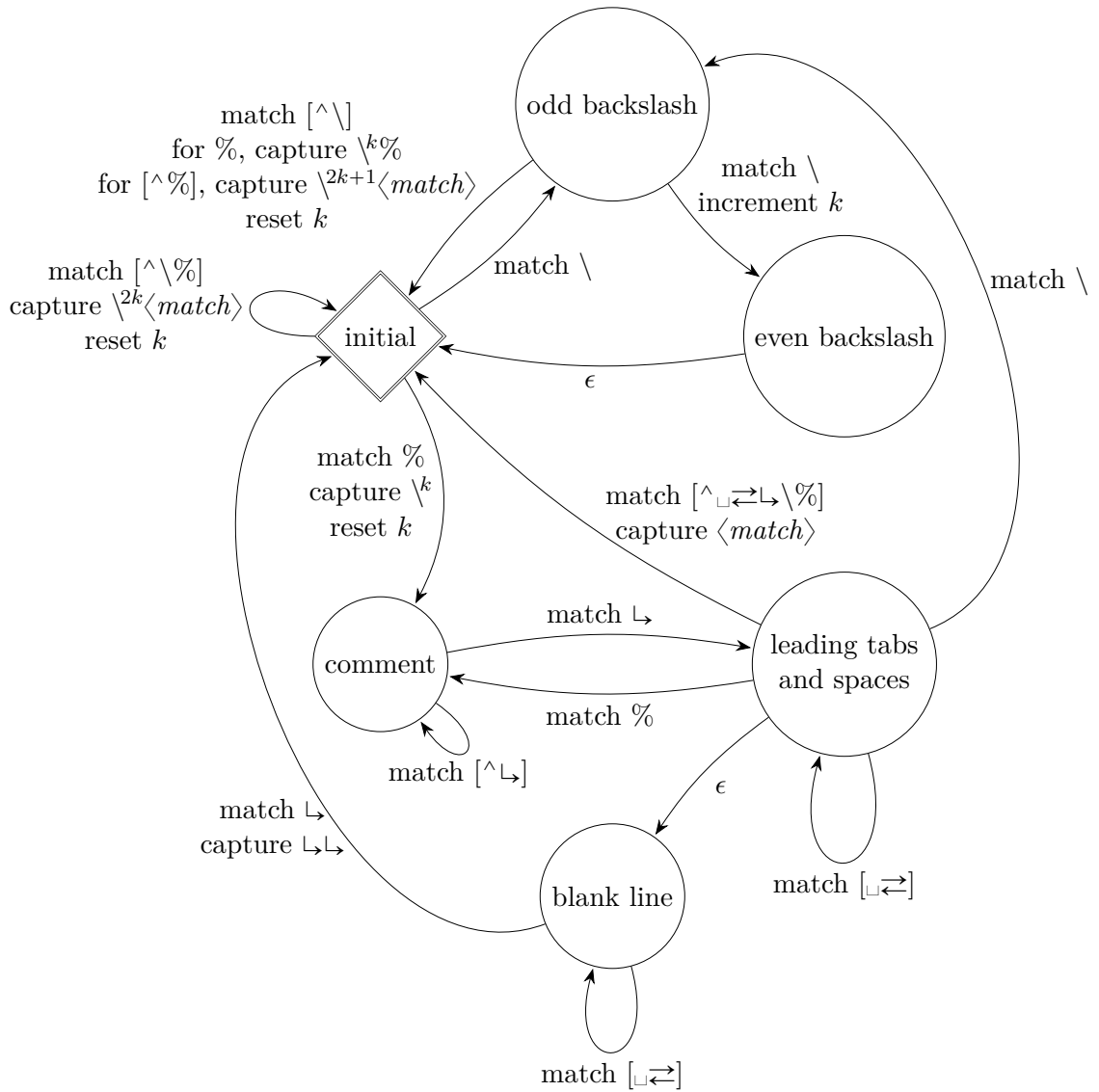
The `parsers.commented_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 6.

```

5136 parsers.commented_line_letter = parsers.linechar
5137 + parsers.newline
5138 - parsers.backslash
5139 - parsers.percent
5140 parsers.commented_line = Cg(Cc(""), "backslashes")
5141 * ((#(parsers.commented_line_letter
5142 - parsers.newline)
5143 * Cb("backslashes")
5144 * Cs(parsers.commented_line_letter
5145 - parsers.newline)^1 -- initial
5146 * Cg(Cc(""), "backslashes"))
5147 + #(parsers.backslash * parsers.backslash)
5148 * Cg((parsers.backslash -- even backslash
5149 * parsers.backslash)^1, "backslashes")
5150 + (parsers.backslash
5151 * (#parsers.percent
5152 * Cb("backslashes")
5153 / function(backslashes)
5154 return string.rep("\\", #backslashes / 2)
5155 end
5156 * C(parsers.percent)
5157 + #parsers.commented_line_letter
5158 * Cb("backslashes")
5159 * Cc("\\")
5160 * C(parsers.commented_line_letter))
5161 * Cg(Cc(""), "backslashes"))^0
5162 * (#parsers.percent
5163 * Cb("backslashes")
5164 / function(backslashes)
5165 return string.rep("\\", #backslashes / 2)
5166 end
5167 * ((parsers.percent -- comment

```





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

```

5168 * parsers.line
5169 * #parsers.blankline) -- blank line
5170 / "\n"
5171 + parsers.percent -- comment
5172 * parsers.line
5173 * parsers.optionalspace) -- leading tabs and spaces
5174 + #(parsers.newline)
5175 * Cb("backslashes")
5176 * C(parsers.newline))
5177
5178 parsers.chunk = parsers.line * (parsers.optionallyindentedline
5179 - parsers.blankline)^0
5180
5181 parsers.attribute_key_char = parsers.alphanumeric + S("_-")
5182 parsers.attribute_key = (parsers.attribute_key_char
5183 - parsers.dash - parsers.digit)
5184 * parsers.attribute_key_char^0
5185 parsers.attribute_value = ((parsers.dquote / "'")
5186 * (parsers.anyescaped - parsers.dquote)^0
5187 * (parsers.dquote / "'"))
5188 + (parsers.anyescaped - parsers.dquote - parsers.rbrace
5189 - parsers.space)^0
5190
5191 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
5192 + C((parsers.hash + parsers.period)
5193 * parsers.attribute_key)
5194 + Cs(parsers.attribute_key
5195 * parsers.optionalspace * parsers.equal * parsers.optionalspace
5196 * parsers.attribute_value)
5197 parsers.attributes = parsers.lbrace
5198 * parsers.optionalspace
5199 * parsers.attribute
5200 * (parsers.spacechar^1
5201 * parsers.attribute)^0
5202 * parsers.optionalspace
5203 * parsers.rbrace
5204
5205 -- block followed by 0 or more optionally
5206 -- indented blocks with first line indented.
5207 parsers.indented_blocks = function(bl)
5208 return Cs(bl
5209 * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
5210 * (parsers.blankline^1 + parsers.eof))
5211 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

5212 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
5213
5214 parsers.bullet = (parsers.bulletchar * #parsers.spacing
5215 * (parsers.tab + parsers.space^-3)
5216 + parsers.space * parsers.bulletchar * #parsers.spacing
5217 * (parsers.tab + parsers.space^-2)
5218 + parsers.space * parsers.space * parsers.bulletchar
5219 * #parsers.spacing
5220 * (parsers.tab + parsers.space^-1)
5221 + parsers.space * parsers.space * parsers.space
5222 * parsers.bulletchar * #parsers.spacing
5223)
5224
5225 local function tickbox(interior)
5226 return parsers.optionalspace * parsers.lbracket
5227 * interior * parsers.rbracket * parsers.spacechar^1
5228 end
5229
5230 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5231 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5232 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
5233

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

5234 parsers.openticks = Cg(parsers.backtick^1, "ticks")
5235
5236 local function captures_equal_length(_,i,a,b)
5237 return #a == #b and i
5238 end
5239
5240 parsers.closeticks = parsers.space^-1
5241 * Cmt(C(parsers.backtick^1)
5242 * Cb("ticks"), captures_equal_length)
5243
5244 parsers.intickschar = (parsers.any - S("\n\r`"))
5245 + (parsers.newline * -parsers.blankline)
5246 + (parsers.space - parsers.closeticks)
5247 + (parsers.backtick^1 - parsers.closeticks)
5248
5249 parsers.inticks = parsers.openticks * parsers.space^-1
5250 * C(parsers.intickschar^0) * parsers.closeticks

```

### 3.1.4.4 Parsers Used for Fenced Code Blocks

```

5251 local function captures_geq_length(_,i,a,b)
5252 return #a >= #b and i
5253 end

```

```

5254
5255 parsers.infostring = (parsers.linechar - (parsers.backtick
5256 + parsers.space^1 * (parsers.newline + parsers.eof)))^0
5257
5258 local fenceindent
5259 parsers.fencehead = function(char)
5260 return C(parsers.nonindentspace) / function(s) fenceindent = #s end
5261 * Cg(char^3, "fencelength")
5262 * parsers.optionalspace * C(parsers.infostring)
5263 * parsers.optionalspace * (parsers.newline + parsers.eof)
5264 end
5265
5266 parsers.fencetail = function(char)
5267 return parsers.nonindentspace
5268 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
5269 * parsers.optionalspace * (parsers.newline + parsers.eof)
5270 + parsers.eof
5271 end
5272
5273 parsers.fencedline = function(char)
5274 return C(parsers.line - parsers.fencetail(char))
5275 / function(s)
5276 local i = 1
5277 local remaining = fenceindent
5278 while true do
5279 local c = s:sub(i, i)
5280 if c == " " and remaining > 0 then
5281 remaining = remaining - 1
5282 i = i + 1
5283 elseif c == "\t" and remaining > 3 then
5284 remaining = remaining - 4
5285 i = i + 1
5286 else
5287 break
5288 end
5289 end
5290 return s:sub(i)
5291 end
5292 end

```

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```

5293 parsers.leader = parsers.space^-3
5294
5295 -- content in balanced brackets, parentheses, or quotes:
5296 parsers.bracketed = P{ parsers.lbracket
5297 * ((parsers.backslash / '"' * parsers.rbracket

```

```

5298 + parsers.any - (parsers.lbracket
5299 + parsers.rbracket
5300 + parsers.blankline^2)
5301) + V(1))^0
5302 * parsers.rbracket }
5303
5304 parsers.inparens = P{ parsers.lparent
5305 * ((parsers.anyescaped - (parsers.lparent
5306 + parsers.rparent
5307 + parsers.blankline^2)
5308) + V(1))^0
5309 * parsers.rparent }
5310
5311 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
5312 * ((parsers.anyescaped - (parsers.squote
5313 + parsers.blankline^2)
5314) + V(1))^0
5315 * parsers.squote }
5316
5317 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
5318 * ((parsers.anyescaped - (parsers.dquote
5319 + parsers.blankline^2)
5320) + V(1))^0
5321 * parsers.dquote }
5322
5323 -- bracketed tag for markdown links, allowing nested brackets:
5324 parsers.tag = parsers.lbracket
5325 * Cs((parsers.alphanumeric^1
5326 + parsers.bracketed
5327 + parsers.inticks
5328 + (parsers.backslash / "\"" * parsers.rbracket
5329 + parsers.any
5330 - (parsers.rbracket + parsers.blankline^2)))^0)
5331 * parsers.rbracket
5332
5333 -- url for markdown links, allowing nested brackets:
5334 parsers.url = parsers.less * Cs((parsers.anyescaped
5335 - parsers.more)^0)
5336 * parsers.more
5337 + Cs((parsers.inparens + (parsers.anyescaped
5338 - parsers.spacing
5339 - parsers.rparent))^1)
5340
5341 -- quoted text, possibly with nested quotes:
5342 parsers.title_s = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5343 + parsers.squoted)^0)
5344 * parsers.squote

```

```

5345
5346 parsers.title_d = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5347 + parsers.dquoted)^0)
5348 * parsers.dquote
5349
5350 parsers.title_p = parsers.lparent
5351 * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5352 * parsers.rparent
5353
5354 parsers.title = parsers.title_d + parsers.title_s + parsers.title_p
5355
5356 parsers.optionaltitle
5357 = parsers.spnl * parsers.title * parsers.spacechar^0
5358 + Cc("")

```

### 3.1.4.6 Parsers Used for HTML

```

5359 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5360 parsers.keyword_exact = function(s)
5361 local parser = P(0)
5362 for i=1,#s do
5363 local c = s:sub(i,i)
5364 local m = c .. upper(c)
5365 parser = parser * S(m)
5366 end
5367 return parser
5368 end
5369
5370 parsers.block_keyword =
5371 parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5372 parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5373 parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5374 parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5375 parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5376 parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5377 parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5378 parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5379 parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5380 parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5381 parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5382 parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5383 parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5384 parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5385 parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5386 parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5387 parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5388 parsers.keyword_exact("td") + parsers.keyword_exact("tr")

```

```

5389
5390 -- There is no reason to support bad html, so we expect quoted attributes
5391 parsers.htmlattributevalue
5392 = parsers.squote * (parsers.any - (parsers.blankline
5393 + parsers.squote))^0
5394 * parsers.squote
5395 + parsers.dquote * (parsers.any - (parsers.blankline
5396 + parsers.dquote))^0
5397 * parsers.dquote
5398
5399 parsers.htmlattribute = parsers.spacing^1
5400 * (parsers.alphanumeric + S("_-"))^1
5401 * parsers.sp * parsers.equal * parsers.sp
5402 * parsers.htmlattributevalue
5403
5404 parsers.htmlcomment = P("<!--")
5405 * parsers.optionalspace
5406 * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5407 * parsers.optionalspace
5408 * P("-->")
5409
5410 parsers.htmlinstruction = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
5411
5412 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5413 * parsers.sp * parsers.more
5414
5415 parsers.openelt_exact = function(s)
5416 return parsers.less * parsers.sp * parsers.keyword_exact(s)
5417 * parsers.htmlattribute^0 * parsers.sp * parsers.more
5418 end
5419
5420 parsers.openelt_block = parsers.sp * parsers.block_keyword
5421 * parsers.htmlattribute^0 * parsers.sp * parsers.more
5422
5423 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5424 * parsers.keyword * parsers.sp * parsers.more
5425
5426 parsers.closeelt_exact = function(s)
5427 return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5428 * parsers.sp * parsers.more
5429 end
5430
5431 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5432 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5433 * parsers.more
5434
5435 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword

```

```

5436 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5437 * parsers.more
5438
5439 parsers.displaytext = (parsers.any - parsers.less)^1
5440
5441 -- return content between two matched HTML tags
5442 parsers.in_matched = function(s)
5443 return { parsers.openelt_exact(s)
5444 * (V(1) + parsers.displaytext
5445 + (parsers.less - parsers.closeelt_exact(s)))^0
5446 * parsers.closeelt_exact(s) }
5447 end
5448
5449 local function parse_matched_tags(s,pos)
5450 local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5451 return lpeg.match(parsers.in_matched(t),s,pos-1)
5452 end
5453
5454 parsers.in_matched_block_tags = parsers.less
5455 * Cmt(#parsers.openelt_block, parse_matched_tags)
5456

```

#### 3.1.4.7 Parsers Used for HTML Entities

```

5457 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5458 * C(parsers.hexdigit^1) * parsers.semicolon
5459 parsers.decentity = parsers.ampersand * parsers.hash
5460 * C(parsers.digit^1) * parsers.semicolon
5461 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5462 * parsers.semicolon

```

#### 3.1.4.8 Helpers for References

```

5463 -- parse a reference definition: [foo]: /bar "title"
5464 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5465 * parsers.spacechar^0 * parsers.url
5466 * parsers.optionaltitle * parsers.blankline^1

```

#### 3.1.4.9 Inline Elements

```

5467 parsers.Inline = V("Inline")
5468 parsers.IndentedInline = V("IndentedInline")
5469
5470 -- parse many p between starter and ender
5471 parsers.between = function(p, starter, ender)
5472 local ender2 = B(parsers.nonspacechar) * ender
5473 return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5474 end

```



```

5475
5476 parsers.urlchar = parsers.anyescaped - parsers.newline - parsers.more

```

### 3.1.4.10 Block Elements

```

5477 parsers.TildeFencedCode
5478 = parsers.fencehead(parsers.tilde)
5479 * Cs(parsers.fencedline(parsers.tilde)^0)
5480 * parsers.fencetail(parsers.tilde)
5481
5482 parsers.BacktickFencedCode
5483 = parsers.fencehead(parsers.backtick)
5484 * Cs(parsers.fencedline(parsers.backtick)^0)
5485 * parsers.fencetail(parsers.backtick)
5486
5487 parsers.lineof = function(c)
5488 return (parsers.leader * (P(c) * parsers.optionalspace)^3
5489 * (parsers.newline * parsers.blankline^1
5490 + parsers.newline^-1 * parsers.eof))
5491 end

```

### 3.1.4.11 Headings

```

5492 -- parse Atx heading start and return level
5493 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
5494 * -parsers.hash / length
5495
5496 -- parse setext header ending and return level
5497 parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5498
5499 local function strip_atx_end(s)
5500 return s:gsub("#%s*\n$", "")
5501 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>`.

```

5502 M.reader = {}
5503 function M.reader.new(writer, options)
5504 local self = {}

```

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

```

5505 self.writer = writer
5506 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.

```

5507 self.parsers = {}
5508 (function(parsers)
5509 setmetatable(self.parsers, {
5510 __index = function (_, key)
5511 return parsers[key]
5512 end
5513 })
5514 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.

```

5515 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.

```

5516 function self.normalize_tag(tag)
5517 return string.lower(
5518 gsub(util.rope_to_string(tag), "[\n\r\t]+", " "))
5519 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.

```

5520 local function iterlines(s, f)
5521 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5522 return util.rope_to_string(rope)
5523 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.

```

5524 if options.preserveTabs then
5525 self.expandtabs = function(s) return s end
5526 else
5527 self.expandtabs = function(s)

```

```

5528 if s:find("\t") then
5529 return iterlines(s, util.expand_tabs_in_line)
5530 else
5531 return s
5532 end
5533 end
5534 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.

```

5535 self.parser_functions = {}
5536 self.create_parser = function(name, grammar, toplevel)
5537 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.

```

5538 if toplevel and options.stripIndent then
5539 local min_prefix_length, min_prefix = nil, ''
5540 str = iterlines(str, function(line)
5541 if lpeg.match(parsers.nonemptyline, line) == nil then
5542 return line
5543 end
5544 line = util.expand_tabs_in_line(line)
5545 local prefix = lpeg.match(C(parsers.optionalspace), line)
5546 local prefix_length = #prefix
5547 local is_shorter = min_prefix_length == nil
5548 is_shorter = is_shorter or prefix_length < min_prefix_length
5549 if is_shorter then
5550 min_prefix_length, min_prefix = prefix_length, prefix
5551 end
5552 return line
5553 end)
5554 str = str:gsub('^' .. min_prefix, '')
5555 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.

```

5556 if toplevel and (options.texComments or options.hybrid) then
5557 str = lpeg.match(Ct(parsers.commented_line^1), str)

```

```

5558 str = util.rope_to_string(str)
5559 end
5560 local res = lpeg.match(grammar(), str)
5561 if res == nil then
5562 error(format("%s failed on:\n%s", name, str:sub(1,20)))
5563 else
5564 return res
5565 end
5566 end
5567 end
5568
5569 self.create_parser("parse_blocks",
5570 function()
5571 return parsers.blocks
5572 end, true)
5573
5574 self.create_parser("parse_blocks_nested",
5575 function()
5576 return parsers.blocks_nested
5577 end, false)
5578
5579 self.create_parser("parse_inlines",
5580 function()
5581 return parsers.inlines
5582 end, false)
5583
5584 self.create_parser("parse_inlines_no_link",
5585 function()
5586 return parsers.inlines_no_link
5587 end, false)
5588
5589 self.create_parser("parse_inlines_no_inline_note",
5590 function()
5591 return parsers.inlines_no_inline_note
5592 end, false)
5593
5594 self.create_parser("parse_inlines_no_html",
5595 function()
5596 return parsers.inlines_no_html
5597 end, false)
5598
5599 self.create_parser("parse_inlines_nbsp",
5600 function()
5601 return parsers.inlines_nbsp
5602 end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```
5603 if options.hashEnumerators then
5604 parsers.dig = parsers.digit + parsers.hash
5605 else
5606 parsers.dig = parsers.digit
5607 end
5608
5609 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5610 + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5611 * (parsers.tab + parsers.space^1)
5612 + C(parsers.dig * parsers.period) * #parsers.spacing
5613 * (parsers.tab + parsers.space^2)
5614 + parsers.space * C(parsers.dig^2 * parsers.period)
5615 * #parsers.spacing
5616 + parsers.space * C(parsers.dig * parsers.period)
5617 * #parsers.spacing
5618 * (parsers.tab + parsers.space^1)
5619 + parsers.space * parsers.space * C(parsers.dig^1
5620 * parsers.period) * #parsers.spacing
```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```
5621 -- strip off leading > and indents, and run through blocks
5622 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^~
5623 1)/""
5624 * parsers.linechar^0 * parsers.newline)^1
5625 * (-(parsers.leader * parsers.more
5626 + parsers.blankline) * parsers.linechar^1
5627 * parsers.newline)^0
5628 if not options.breakableBlockquotes then
5629 parsers.blockquote_body = parsers.blockquote_body
5630 * (parsers.blankline^0 / "")
5631 end
```

### 3.1.5.5 Parsers Used for Notes (local)

#### 3.1.5.6 Helpers for Links and References (local)

```
5632 -- List of references defined in the document
5633 local references
5634
5635 -- add a reference to the list
5636 local function register_link(tag,url,title)
5637 references[self.normalize_tag(tag)] = { url = url, title = title }
5638 return ""
5639 end
```

```

5640
5641 -- lookup link reference and return either
5642 -- the link or nil and fallback text.
5643 local function lookup_reference(label,sps,tag)
5644 local tagpart
5645 if not tag then
5646 tag = label
5647 tagpart = ""
5648 elseif tag == "" then
5649 tag = label
5650 tagpart = "[]"
5651 else
5652 tagpart = {"[",
5653 self.parser_functions.parse_inlines(tag),
5654 "]" }
5655 end
5656 if sps then
5657 tagpart = {sps, tagpart}
5658 end
5659 local r = references[self.normalize_tag(tag)]
5660 if r then
5661 return r
5662 else
5663 return nil, {"[",
5664 self.parser_functions.parse_inlines(label),
5665 "]", tagpart}
5666 end
5667 end
5668
5669 -- lookup link reference and return a link, if the reference is found,
5670 -- or a bracketed label otherwise.
5671 local function indirect_link(label,sps,tag)
5672 return writer.defer_call(function()
5673 local r,fallback = lookup_reference(label,sps,tag)
5674 if r then
5675 return writer.link(
5676 self.parser_functions.parse_inlines_no_link(label),
5677 r.url, r.title)
5678 else
5679 return fallback
5680 end
5681 end)
5682 end
5683
5684 -- lookup image reference and return an image, if the reference is found,
5685 -- or a bracketed label otherwise.
5686 local function indirect_image(label,sps,tag)

```

```

5687 return writer.defer_call(function()
5688 local r, fallback = lookup_reference(label, sps, tag)
5689 if r then
5690 return writer.image(writer.string(label), r.url, r.title)
5691 else
5692 return {"!", fallback}
5693 end
5694 end)
5695 end

```

### 3.1.5.7 Inline Elements (local)

```

5696 parsers.Str = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5697 / writer.string
5698
5699 parsers.Symbol = (V("SpecialChar") - parsers.tightblocksep)
5700 / writer.string
5701
5702 parsers.Ellipsis = P("...") / writer.ellipsis
5703
5704 parsers.Smart = parsers.Ellipsis
5705
5706 parsers.Code = parsers.inticks / writer.code
5707
5708 if options.blankBeforeBlockquote then
5709 parsers.bqstart = parsers.fail
5710 else
5711 parsers.bqstart = parsers.more
5712 end
5713
5714 if options.blankBeforeHeading then
5715 parsers.headerstart = parsers.fail
5716 else
5717 parsers.headerstart = parsers.hash
5718 + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5719 * parsers.optionalspace * parsers.newline)
5720 end
5721
5722 parsers.EndlineExceptions
5723 = parsers.blankline -- paragraph break
5724 + parsers.tightblocksep -- nested list
5725 + parsers.eof -- end of document
5726 + parsers.bqstart
5727 + parsers.headerstart
5728
5729 parsers.Endline = parsers.newline
5730 * -V("EndlineExceptions")

```

```

5731 * parsers.spacechar^0
5732 / (options.hardLineBreaks and writer.linebreak
5733 or writer.space)
5734
5735 parsers.OptionalIndent
5736 = parsers.spacechar^1 / writer.space
5737
5738 parsers.Space
5739 = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5740 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5741 + parsers.spacechar^1 * parsers.Endline
5742 * parsers.optionalspace
5743 / (options.hardLineBreaks
5744 and writer.linebreak
5745 or writer.space)
5746 + parsers.spacechar^1 * parsers.optionalspace
5747 / writer.space
5748
5749 parsers.NonbreakingEndline
5750 = parsers.newline
5751 * -V("EndlineExceptions")
5752 * parsers.spacechar^0
5753 / (options.hardLineBreaks and writer.linebreak
5754 or writer.nbsp)
5755
5756 parsers.NonbreakingSpace
5757 = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5758 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5759 + parsers.spacechar^1 * parsers.Endline
5760 * parsers.optionalspace
5761 / (options.hardLineBreaks
5762 and writer.linebreak
5763 or writer.nbsp)
5764 + parsers.spacechar^1 * parsers.optionalspace
5765 / writer.nbsp
5766
5767 if options.underscores then
5768 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
5769 parsers.doubleasterisks)
5770 + parsers.between(parsers.Inline, parsers.doubleunderscores,
5771 parsers.doubleunderscores)
5772) / writer.strong
5773
5774 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
5775 parsers.asterisk)
5776 + parsers.between(parsers.Inline, parsers.underscore,
5777 parsers.underscore)
5778) / writer.emphasis

```



```

5778 else
5779 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
5780 parsers.doubleasterisks)
5781) / writer.strong
5782
5783 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
5784 parsers.asterisk)
5785) / writer.emphasis
5786 end
5787
5788 parsers.AutoLinkUrl = parsers.less
5789 * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5790 * parsers.more
5791 / function(url)
5792 return writer.link(writer.escape(url), url)
5793 end
5794
5795 parsers.AutoLinkEmail = parsers.less
5796 * C((parsers.alphanumeric + S("-._+"))^1
5797 * P("@") * parsers.urlchar^1)
5798 * parsers.more
5799 / function(email)
5800 return writer.link(writer.escape(email),
5801 "mailto:".email)
5802 end
5803
5804 parsers.AutoLinkRelativeReference
5805 = parsers.less
5806 * C(parsers.urlchar^1)
5807 * parsers.more
5808 / function(url)
5809 return writer.link(writer.escape(url), url)
5810 end
5811
5812 parsers.DirectLink = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5813 * parsers.spnl
5814 * parsers.lparent
5815 * (parsers.url + C("")) -- link can be empty [foo]()
5816 * parsers.optionaltitle
5817 * parsers.rparent
5818 / writer.link
5819
5820 parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
5821 1
5822 / indirect_link
5823 -- parse a link or image (direct or indirect)

```

```

5824 parsers.Link = parsers.DirectLink + parsers.IndirectLink
5825
5826 parsers.DirectImage = parsers.exclamation
5827 * (parsers.tag / self.parser_functions.parse_inlines)
5828 * parsers.spnl
5829 * parsers.lparent
5830 * (parsers.url + Cc("")) -- link can be empty [foo]()
5831 * parsers.optionaltitle
5832 * parsers.rparent
5833 / writer.image
5834
5835 parsers.IndirectImage = parsers.exclamation * parsers.tag
5836 * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5837
5838 parsers.Image = parsers.DirectImage + parsers.IndirectImage
5839
5840 -- avoid parsing long strings of * or _ as emph/strong
5841 parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
5842 / writer.string
5843
5844 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
5845
5846 parsers.InlineHtml = parsers.emptyelt_any / writer.inline_html_tag
5847 + (parsers.htmlcomment / self.parser_functions.parse_inlines_no
5848 / writer.inline_html_comment
5849 + parsers.htmlinstruction
5850 + parsers.openelt_any / writer.inline_html_tag
5851 + parsers.closeelt_any / writer.inline_html_tag
5852
5853 parsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
5854 + parsers.decentity / entities.dec_entity / writer.string
5855 + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.8 Block Elements (local)

```

5856 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_nest
5857 / writer.block_html_comment
5858 + parsers.emptyelt_block / writer.block_html_element
5859 + parsers.openelt_exact("hr") / writer.block_html_element
5860 + parsers.in_matched_block_tags / writer.block_html_element
5861 + parsers.htmlinstruction
5862
5863 parsers.Verbatim = Cs((parsers.blanklines
5864 * ((parsers.indentedline - parsers.blankline))^1)^1
5865) / self.expandtabs / writer.verbatim
5866
5867 parsers.Blockquote = Cs(parsers.blockquote_body^1)

```

```

5868 / self.parser_functions.parse_blocks_nested
5869 / writer.blockquote
5870
5871 parsers.ThematicBreak = (parsers.lineof(parsers.asterisk)
5872 + parsers.lineof(parsers.dash)
5873 + parsers.lineof(parsers.underscore)
5874) / writer.thematic_break
5875
5876 parsers.Reference = parsers.define_reference_parser / register_link
5877
5878 parsers.Paragraph = parsers.nonindent_space * Ct(parsers.Inline^1)
5879 * (parsers.newline
5880 * (parsers.blankline^1
5881 + #parsers.hash
5882 + #(parsers.leader * parsers.more * parsers.space^-
5883 + parsers.eof
5884)
5885 + parsers.eof)
5886 / writer.paragraph
5887
5888 parsers.Plain = parsers.nonindent_space * Ct(parsers.Inline^1)
5889 / writer.plain

```

### 3.1.5.9 Lists (local)

```

5890 parsers.starter = parsers.bullet + parsers.enumerator
5891
5892 if options.taskLists then
5893 parsers.tickbox = (parsers.ticked_box
5894 + parsers.halfticked_box
5895 + parsers.unticked_box
5896) / writer.tickbox
5897 else
5898 parsers.tickbox = parsers.fail
5899 end
5900
5901 -- we use \001 as a separator between a tight list item and a
5902 -- nested list under it.
5903 parsers.NestedList = Cs((parsers.optionallyindentedline
5904 - parsers.starter)^1)
5905 / function(a) return "\001"..a end
5906
5907 parsers.ListBlockLine = parsers.optionallyindentedline
5908 - parsers.blankline - (parsers.indent^-1
5909 * parsers.starter)
5910

```

```

5911 parsers.ListBlock = parsers.line * parsers.ListBlockLine^0
5912
5913 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5914 * parsers.ListBlock
5915
5916 parsers.TightListItem = function(starter)
5917 return -parsers.ThematicBreak
5918 * (Cs(starter / "" * parsers.checkbox^-1 * parsers.ListBlock * parsers.Nested
5919 1)
5919 / self.parser_functions.parse_blocks_nested)
5920 * -(parsers.blanklines * parsers.indent)
5921 end
5922
5923 parsers.LooseListItem = function(starter)
5924 return -parsers.ThematicBreak
5925 * Cs(starter / "" * parsers.checkbox^-1 * parsers.ListBlock * Cc("\n")
5926 * (parsers.NestedList + parsers.ListContinuationBlock^0)
5927 * (parsers.blanklines / "\n\n")
5928) / self.parser_functions.parse_blocks_nested
5929 end
5930
5931 parsers.BulletList = (Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
5932 * parsers.skipblanklines * -parsers.bullet
5933 + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5934 * parsers.skipblanklines)
5935 / writer.bulletlist
5936
5937 local function ordered_list(items,tight,startnum)
5938 if options.startNumber then
5939 startnum = tonumber(startnum) or 1 -- fallback for '#'
5940 if startnum ~= nil then
5941 startnum = math.floor(startnum)
5942 end
5943 else
5944 startnum = nil
5945 end
5946 return writer.orderedlist(items,tight,startnum)
5947 end
5948
5949 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
5950 (Ct(parsers.TightListItem(Cb("listtype")))
5951 * parsers.TightListItem(parsers.enumerator)^0)
5952 * Cc(true) * parsers.skipblanklines * -parsers.enumerator
5953 + Ct(parsers.LooseListItem(Cb("listtype")))
5954 * parsers.LooseListItem(parsers.enumerator)^0)
5955 * Cc(false) * parsers.skipblanklines
5956) * Cb("listtype") / ordered_list

```

### 3.1.5.10 Blank (local)

```
5957 parsers.Blank = parsers.blankline / ""
5958 + parsers.Reference
5959 + (parsers.tightblocksep / "\n")
```

### 3.1.5.11 Headings (local)

```
5960 -- parse atx header
5961 parsers.AtxHeading = Cg(parsers.heading_start, "level")
5962 * parsers.optionalspace
5963 * (C(parsers.line)
5964 / strip_atx_end
5965 / self.parser_functions.parse_inlines)
5966 * Cb("level")
5967 / writer.heading
5968
5969 parsers.SetextHeading = #(parsers.line * S("--"))
5970 * Ct(parsers.linechar^1
5971 / self.parser_functions.parse_inlines)
5972 * parsers.newline
5973 * parsers.heading_level
5974 * parsers.optionalspace
5975 * parsers.newline
5976 / writer.heading
5977
5978 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  $\text{\TeX}$  output.

```
5979 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.

```
5980 local walkable_syntax = (function(global_walkable_syntax)
5981 local local_walkable_syntax = {}
5982 for lhs, rule in pairs(global_walkable_syntax) do
5983 local_walkable_syntax[lhs] = util.table_copy(rule)
5984 end
5985 return local_walkable_syntax
5986 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.

```

5987 local current_extension_name = nil
5988 self.insert_pattern = function(selector, pattern, pattern_name)
5989 assert(pattern_name == nil or type(pattern_name) == "string")
5990 local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
5991 assert(lhs ~= nil,
5992 [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
5993 .. selector .. ["])
5994 assert(walkable_syntax[lhs] ~= nil,
5995 [[Rule]] .. lhs .. [[-> ... does not exist in markdown grammar]])
5996 assert(pos == "before" or pos == "after" or pos == "instead of",
5997 [[Expected positional specifier "before", "after", or "instead of", not "]]
5998 .. pos .. ["])
5999 local rule = walkable_syntax[lhs]
6000 local index = nil
6001 for current_index, current_rhs in ipairs(rule) do
6002 if type(current_rhs) == "string" and current_rhs == rhs then
6003 index = current_index
6004 if pos == "after" then
6005 index = index + 1
6006 end
6007 break
6008 end
6009 end
6010 assert(index ~= nil,
6011 [[Rule]] .. lhs .. [[->]] .. rhs
6012 .. [[does not exist in markdown grammar]])
6013 local accountable_pattern
6014 if current_extension_name then
6015 accountable_pattern = { pattern, current_extension_name, pattern_name }
6016 else
6017 assert(type(pattern) == "string",
6018 [[reader->insert_pattern() was called outside an extension with]]
6019 .. [[a PEG pattern instead of a rule name]])
6020 accountable_pattern = pattern
6021 end
6022 if pos == "instead of" then
6023 rule[index] = accountable_pattern
6024 else
6025 table.insert(rule, index, accountable_pattern)
6026 end
6027 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.

```

6028 local syntax =
6029 { "Blocks",
6030
6031 Blocks = (V("ExpectedJekyllData")
6032 * (V("Blank")^0 / writer.interblocksep)
6033)^-1
6034 * V("Blank")^0
6035 * V("Block")^-1
6036 * (V("Blank")^0 / writer.interblocksep
6037 * V("Block"))^0
6038 * V("Blank")^0 * parsers.eof,
6039
6040 ExpectedJekyllData = parsers.fail,
6041
6042 Blank = parsers.Blank,
6043
6044 Blockquote = parsers.Blockquote,
6045 Verbatim = parsers.Verbatim,
6046 ThematicBreak = parsers.ThematicBreak,
6047 BulletList = parsers.BulletList,
6048 OrderedList = parsers.OrderedList,
6049 Heading = parsers.Heading,
6050 DisplayHtml = parsers.DisplayHtml,
6051 Paragraph = parsers.Paragraph,
6052 Plain = parsers.Plain,
6053 EndlineExceptions = parsers.EndlineExceptions,
6054
6055 Str = parsers.Str,
6056 Space = parsers.Space,
6057 OptionalIndent = parsers.OptionalIndent,
6058 Endline = parsers.Endline,
6059 U1OrStarLine = parsers.U1OrStarLine,
6060 Strong = parsers.Strong,
6061 Emph = parsers.Emph,
6062 Link = parsers.Link,
6063 Image = parsers.Image,
6064 Code = parsers.Code,
6065 AutoLinkUrl = parsers.AutoLinkUrl,
6066 AutoLinkEmail = parsers.AutoLinkEmail,
6067 AutoLinkRelativeReference
6068 = parsers.AutoLinkRelativeReference,
6069 InlineHtml = parsers.InlineHtml,
6070 HtmlEntity = parsers.HtmlEntity,
6071 EscapedChar = parsers.EscapedChar,
6072 Smart = parsers.Smart,
6073 Symbol = parsers.Symbol,
6074 SpecialChar = parsers.fail,

```

```
6075 }
```

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.

```
6076 self.update_rule = function(rule_name, pattern)
6077 assert(current_extension_name ~= nil)
6078 assert(syntax[rule_name] ~= nil,
6079 [[Rule]] .. rule_name .. [[-> ... does not exist in markdown grammar]])
6080 local accountable_pattern = { pattern, current_extension_name, rule_name }
6081 walkable_syntax[rule_name] = { accountable_pattern }
6082 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.

```
6083 local special_characters = {}
6084 self.add_special_character = function(c)
6085 table.insert(special_characters, c)
6086 syntax.SpecialChar = S(table.concat(special_characters, ""))
6087 end
6088
6089 self.add_special_character("*")
6090 self.add_special_character("`")
6091 self.add_special_character("[")
6092 self.add_special_character("]")
6093 self.add_special_character("<")
6094 self.add_special_character("!")
6095 self.add_special_character("\\")
```

Apply syntax extensions.

```
6096 for _, extension in ipairs(extensions) do
6097 current_extension_name = extension.name
6098 extension.extend_writer(writer)
6099 extension.extend_reader(self)
6100 end
6101 current_extension_name = nil
```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```
6102 if options.debugExtensions then
6103 local sorted_lhs = {}
6104 for lhs, _ in pairs(walkable_syntax) do
6105 table.insert(sorted_lhs, lhs)
6106 end
6107 table.sort(sorted_lhs)
6108
6109 local output_lines = {"{"}
```



```

6110 for lhs_index, lhs in ipairs(sorted_lhs) do
6111 local encoded_lhs = util.encode_json_string(lhs)
6112 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])
6113 local rule = walkable_syntax[lhs]
6114 for rhs_index, rhs in ipairs(rule) do
6115 local human_readable_rhs
6116 if type(rhs) == "string" then
6117 human_readable_rhs = rhs
6118 else
6119 local pattern_name
6120 if rhs[3] then
6121 pattern_name = rhs[3]
6122 else
6123 pattern_name = "Anonymous Pattern"
6124 end
6125 local extension_name = rhs[2]
6126 human_readable_rhs = pattern_name .. [[(]] .. extension_name .. [[)]]
6127 end
6128 local encoded_rhs = util.encode_json_string(human_readable_rhs)
6129 local output_line = [[]] .. encoded_rhs
6130 if rhs_index < #rule then
6131 output_line = output_line .. ", "
6132 end
6133 table.insert(output_lines, output_line)
6134 end
6135 local output_line = "]"
6136 if lhs_index < #sorted_lhs then
6137 output_line = output_line .. ", "
6138 end
6139 table.insert(output_lines, output_line)
6140 end
6141 table.insert(output_lines, "}")
6142
6143 local output = table.concat(output_lines, "\n")
6144 local output_filename = options.debugExtensionsFileName
6145 local output_file = assert(io.open(output_filename, "w"),
6146 [[Could not open file]] .. output_filename .. [[for writing]])
6147 assert(output_file:write(output))
6148 assert(output_file:close())
6149 end

```

Duplicate the [Inline](#) rule as [IndentedInline](#) with the right-hand-side terminal symbol [Space](#) replaced with [OptionalIndent](#).

```

6150 walkable_syntax["IndentedInline"] = util.table_copy(
6151 walkable_syntax["Inline"])
6152 self.insert_pattern(
6153 "IndentedInline instead of Space",

```

```
6154 "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`.

```
6155 for lhs, rule in pairs(walkable_syntax) do
6156 syntax[lhs] = parsers.fail
6157 for _, rhs in ipairs(rule) do
6158 local pattern
```

Although the interface of the `reader->insert_pattern` method does document this (see Section 2.1.2), 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.

```
6159 if type(rhs) == "string" then
6160 pattern = V(rhs)
6161 else
6162 pattern = rhs[1]
6163 if type(pattern) == "string" then
6164 pattern = V(pattern)
6165 end
6166 end
6167 syntax[lhs] = syntax[lhs] + pattern
6168 end
6169 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.

```
6170 if options.underscores then
6171 self.add_special_character("_")
6172 end
6173
6174 if not options.codeSpans then
6175 syntax.Code = parsers.fail
6176 end
6177
6178 if not options.html then
6179 syntax.DisplayHtml = parsers.fail
6180 syntax.InlineHtml = parsers.fail
6181 syntax.HtmlEntity = parsers.fail
6182 else
6183 self.add_special_character("&")
6184 end
6185
6186 if options.preserveTabs then
6187 options.stripIndent = false
```

```

6188 end
6189
6190 if not options.smartEllipses then
6191 syntax.Smart = parsers.fail
6192 else
6193 self.add_special_character(".")
6194 end
6195
6196 if not options.relativeReferences then
6197 syntax.AutoLinkRelativeReference = parsers.fail
6198 end
6199
6200 local blocks_nested_t = util.table_copy(syntax)
6201 blocks_nested_t.ExpectedJekyllData = parsers.fail
6202 parsers.blocks_nested = Ct(blocks_nested_t)
6203
6204 parsers.blocks = Ct(syntax)
6205
6206 local inlines_t = util.table_copy(syntax)
6207 inlines_t[1] = "Inlines"
6208 inlines_t.Inlines = parsers.Inline~0 * (parsers.spacing~0 * parsers.eof / "")
6209 parsers.inlines = Ct(inlines_t)
6210
6211 local inlines_no_link_t = util.table_copy(inlines_t)
6212 inlines_no_link_t.Link = parsers.fail
6213 parsers.inlines_no_link = Ct(inlines_no_link_t)
6214
6215 local inlines_no_inline_note_t = util.table_copy(inlines_t)
6216 inlines_no_inline_note_t.InlineNote = parsers.fail
6217 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6218
6219 local inlines_no_html_t = util.table_copy(inlines_t)
6220 inlines_no_html_t.DisplayHtml = parsers.fail
6221 inlines_no_html_t.InlineHtml = parsers.fail
6222 inlines_no_html_t.HtmlEntity = parsers.fail
6223 parsers.inlines_no_html = Ct(inlines_no_html_t)
6224
6225 local inlines_nbsp_t = util.table_copy(inlines_t)
6226 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6227 inlines_nbsp_t.Space = parsers.NonbreakingSpace
6228 parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

Return a function that converts markdown string `input` into a plain  $\TeX$  output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

6229 return function(input)
6230 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.

```

6231 local opt_string = {}
6232 for k, _ in pairs(defaultOptions) do
6233 local v = options[k]
6234 if type(v) == "table" then
6235 for _, i in ipairs(v) do
6236 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
6237 end
6238 elseif k ~= "cacheDir" then
6239 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6240 end
6241 end
6242 table.sort(opt_string)
6243 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
6244 local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output via the `writer->pack` method.

```

6245 local function convert(input)
6246 local document = self.parser_functions.parse_blocks(input)
6247 return util.rope_to_string(writer.document(document))
6248 end
6249 if options.eagerCache or options.finalizeCache then
6250 local name = util.cache(options.cacheDir, input, salt, convert,
6251 ".md" .. writer.suffix)
6252 output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

6253 else
6254 output = convert(input)
6255 end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

6256 if options.finalizeCache then
6257 local file, mode
6258 if options.frozenCacheCounter > 0 then
6259 mode = "a"
6260 else
6261 mode = "w"
6262 end
6263 file = assert(io.open(options.frozenCacheFileName, mode),
6264 [[Could not open file]] .. options.frozenCacheFileName
6265 .. [[for writing]])
6266 assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname]]
6267 .. [[markdownFrozenCache]] .. options.frozenCacheCounter

```

```

6268 .. [[\endcsname{}]] .. output .. [[]] .. "\n"))
6269 assert(file:close())
6270 end
6271 return output
6272 end
6273 end
6274 return self
6275 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.

```

6276 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.

```

6277 M.extensions.citations = function(citation_nbsps)

```

Define table `escaped_citation_chars` containing the characters to escape in citations.

```

6278 local escaped_citation_chars = {
6279 [{""] = "\\markdownRendererLeftBrace{}",
6280 ["}"] = "\\markdownRendererRightBrace{}",
6281 [%"] = "\\markdownRendererPercentSign{}",
6282 ["\\"] = "\\markdownRendererBackslash{}",
6283 [#"] = "\\markdownRendererHash{}",
6284 }
6285 return {
6286 name = "built-in citations syntax extension",
6287 extend_writer = function(self)
6288 local options = self.options
6289

```

Use the `escaped_citation_chars` to create the `escape_citation` escaper functions.

```

6290 local escape_citation = util.escaper(
6291 escaped_citation_chars,
6292 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.

```
6293 if options.hybrid then
6294 self.citation = self.escape_minimal
6295 else
6296 self.citation = escape_citation
6297 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.

```
6298 function self.citations(text_cites, cites)
6299 local buffer = {"\markdownRenderer", text_cites and "TextCite" or "Cite",
6300 "{", #cites, "}"}
6301 for _,cite in ipairs(cites) do
6302 buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6303 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
6304 end
6305 return buffer
6306 end
6307 end, extend_reader = function(self)
6308 local parsers = self.parsers
6309 local writer = self.writer
6310
6311 local citation_chars
6312 = parsers.alphanumeric
6313 + S("#$%&-+<>~/_")
6314
6315 local citation_name
6316 = Cs(parsers.dash^-1) * parsers.at
6317 * Cs(citation_chars
6318 * (((citation_chars + parsers.internal_punctuation
6319 - parsers.comma - parsers.semicolon)
```

```

6320 * -#((parsers.internal_punctuation - parsers.comma
6321 - parsers.semicolon)^0
6322 * -(citation_chars + parsers.internal_punctuation
6323 - parsers.comma - parsers.semicolon))^0
6324 * citation_chars)^-1)
6325
6326 local citation_body_prenote
6327 = Cs((parsers.alphanumeric^1
6328 + parsers.bracketed
6329 + parsers.inticks
6330 + (parsers.anyescaped
6331 - (parsers.rbracket + parsers.blankline^2))
6332 - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6333
6334 local citation_body_postnote
6335 = Cs((parsers.alphanumeric^1
6336 + parsers.bracketed
6337 + parsers.inticks
6338 + (parsers.anyescaped
6339 - (parsers.rbracket + parsers.semicolon
6340 + parsers.blankline^2))
6341 - (parsers.spnl * parsers.rbracket))^0)
6342
6343 local citation_body_chunk
6344 = citation_body_prenote
6345 * parsers.spnl * citation_name
6346 * (parsers.internal_punctuation - parsers.semicolon)^-
1
6347 * parsers.spnl * citation_body_postnote
6348
6349 local citation_body
6350 = citation_body_chunk
6351 * (parsers.semicolon * parsers.spnl
6352 * citation_body_chunk)^0
6353
6354 local citation_headless_body_postnote
6355 = Cs((parsers.alphanumeric^1
6356 + parsers.bracketed
6357 + parsers.inticks
6358 + (parsers.anyescaped
6359 - (parsers.rbracket + parsers.at
6360 + parsers.semicolon + parsers.blankline^2))
6361 - (parsers.spnl * parsers.rbracket))^0)
6362
6363 local citation_headless_body
6364 = citation_headless_body_postnote
6365 * (parsers.sp * parsers.semicolon * parsers.spnl

```

```

6366 * citation_body_chunk)^0
6367
6368 local citations
6369 = function(text_cites, raw_cites)
6370 local function normalize(str)
6371 if str == "" then
6372 str = nil
6373 else
6374 str = (citation_nbsps and
6375 self.parser_functions.parse_inlines_nbsp or
6376 self.parser_functions.parse_inlines)(str)
6377 end
6378 return str
6379 end
6380
6381 local cites = {}
6382 for i = 1,#raw_cites,4 do
6383 cites[#cites+1] = {
6384 prenote = normalize(raw_cites[i]),
6385 suppress_author = raw_cites[i+1] == "-",
6386 name = writer.citation(raw_cites[i+2]),
6387 postnote = normalize(raw_cites[i+3]),
6388 }
6389 end
6390 return writer.citations(text_cites, cites)
6391 end
6392
6393 local TextCitations
6394 = Ct((parsers.spnl
6395 * Cc("")
6396 * citation_name
6397 * ((parsers.spnl
6398 * parsers.lbracket
6399 * citation_headless_body
6400 * parsers.rbracket) + Cc("")))^1)
6401 / function(raw_cites)
6402 return citations(true, raw_cites)
6403 end
6404
6405 local ParenthesizedCitations
6406 = Ct((parsers.spnl
6407 * parsers.lbracket
6408 * citation_body
6409 * parsers.rbracket)^1)
6410 / function(raw_cites)
6411 return citations(false, raw_cites)
6412 end

```



```

6413
6414 local Citations = TextCitations + ParenthesizedCitations
6415
6416 self.insert_pattern("Inline after Emph",
6417 Citations, "Citations")
6418
6419 self.add_special_character("@")
6420 self.add_special_character("-")
6421 end
6422 }
6423 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.

```

6424 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.

```

6425 local languages_json = (function()
6426 local base, prev, curr
6427 for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
6428 local file = io.open(pathname, "r")
6429 if not file then goto continue end
6430 local input = assert(file:read("*a"))
6431 assert(file:close())
6432 local json = input:gsub('"[^\\n]-"', "[%1]='')
6433 curr = load("_ENV = {}; return "..json")()
6434 if type(curr) == "table" then
6435 if base == nil then
6436 base = curr
6437 else
6438 setmetatable(prev, { __index = curr })
6439 end
6440 prev = curr
6441 end
6442 ::continue::
6443 end
6444 return base or {}
6445 end)()
6446
6447 return {
6448 name = "built-in content_blocks syntax extension",

```

```
6449 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.

```
6450 function self.contentblock(src,suf,type,tit)
6451 if not self.is_writing then return "" end
6452 src = src..".."..suf
6453 suf = suf:lower()
6454 if type == "onlineimage" then
6455 return {"\markdownRendererContentBlockOnlineImage{" ,suf,"}"} ,
6456 {"",self.string(src),""} ,
6457 {"",self.uri(src),""} ,
6458 {"",self.string(tit or ""),""}
6459 elseif languages_json[suf] then
6460 return {"\markdownRendererContentBlockCode{" ,suf,"}"} ,
6461 {"",self.string(languages_json[suf]),""} ,
6462 {"",self.string(src),""} ,
6463 {"",self.uri(src),""} ,
6464 {"",self.string(tit or ""),""}
6465 else
6466 return {"\markdownRendererContentBlock{" ,suf,"}"} ,
6467 {"",self.string(src),""} ,
6468 {"",self.uri(src),""} ,
6469 {"",self.string(tit or ""),""}
6470 end
6471 end
6472 end, extend_reader = function(self)
6473 local parsers = self.parsers
6474 local writer = self.writer
6475
6476 local contentblock_tail
6477 = parsers.optionaltitle
6478 * (parsers.newline + parsers.eof)
6479
6480 -- case insensitive online image suffix:
6481 local onlineimagesuffix
6482 = (function(...)
6483 local parser = nil
6484 for _, suffix in ipairs({...}) do
6485 local pattern=nil
6486 for i=1,#suffix do
6487 local char=suffix:sub(i,i)
6488 char = S(char:lower()..char:upper())
6489 if pattern == nil then
6490 pattern = char
```

```

6491 else
6492 pattern = pattern * char
6493 end
6494 end
6495 if parser == nil then
6496 parser = pattern
6497 else
6498 parser = parser + pattern
6499 end
6500 end
6501 return parser
6502 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6503
6504 -- online image url for iA Writer content blocks with mandatory suffix,
6505 -- allowing nested brackets:
6506 local onlineimageurl
6507 = (parsers.less
6508 * Cs((parsers.anyescaped
6509 - parsers.more
6510 - #(parsers.period
6511 * onlineimagesuffix
6512 * parsers.more
6513 * contentblock_tail))^0)
6514 * parsers.period
6515 * Cs(onlineimagesuffix)
6516 * parsers.more
6517 + (Cs((parsers.inparens
6518 + (parsers.anyescaped
6519 - parsers.spacing
6520 - parsers.rparent
6521 - #(parsers.period
6522 * onlineimagesuffix
6523 * contentblock_tail))))^0)
6524 * parsers.period
6525 * Cs(onlineimagesuffix))
6526) * Cc("onlineimage")
6527
6528 -- filename for iA Writer content blocks with mandatory suffix:
6529 local localfilepath
6530 = parsers.slash
6531 * Cs((parsers.anyescaped
6532 - parsers.tab
6533 - parsers.newline
6534 - #(parsers.period
6535 * parsers.alphanumeric^1
6536 * contentblock_tail))^1)
6537 * parsers.period

```

```

6538 * Cs(parsers.alphanumeric^1)
6539 * Cc("localfile")
6540
6541 local ContentBlock
6542 = parsers.leader
6543 * (localfilepath + onlineimageurl)
6544 * contentblock_tail
6545 / writer.contentblock
6546
6547 self.insert_pattern("Block before Blockquote",
6548 ContentBlock, "ContentBlock")
6549 end
6550 }
6551 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.

```

6552 M.extensions.definition_lists = function(tight_lists)
6553 return {
6554 name = "built-in definition_lists syntax extension",
6555 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.

```

6556 local function dlitem(term, defs)
6557 local retVal = {"\\markdownRendererDlItem{" ,term,""}
6558 for _, def in ipairs(defs) do
6559 retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
6560 "\\markdownRendererDlDefinitionEnd "}
6561 end
6562 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
6563 return retVal
6564 end
6565
6566 function self.definitionlist(items,tight)
6567 if not self.is_writing then return "" end
6568 local buffer = {}
6569 for _,item in ipairs(items) do
6570 buffer[#buffer + 1] = dlitem(item.term, item.definitions)
6571 end
6572 if tight and tight_lists then
6573 return {"\\markdownRendererDlBeginTight\\n", buffer,
6574 "\\n\\markdownRendererDlEndTight"}

```

```

6575 else
6576 return {"\\markdownRendererDlBegin\n", buffer,
6577 "\\n\\markdownRendererDlEnd"}
6578 end
6579 end
6580 end, extend_reader = function(self)
6581 local parsers = self.parsers
6582 local writer = self.writer
6583
6584 local defstartchar = S("~:")
6585
6586 local defstart = (defstartchar * #parsers.spacing
6587 * (parsers.tab + parsers.space^-3)
6588 + parsers.space * defstartchar * #parsers.spacing
6589 * (parsers.tab + parsers.space^-2)
6590 + parsers.space * parsers.space * defstartchar
6591 * #parsers.spacing
6592 * (parsers.tab + parsers.space^-1)
6593 + parsers.space * parsers.space * parsers.space
6594 * defstartchar * #parsers.spacing
6595)
6596
6597 local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
6598
6599 local function definition_list_item(term, defs, _)
6600 return { term = self.parser_functions.parse_inlines(term),
6601 definitions = defs }
6602 end
6603
6604 local DefinitionListItemLoose
6605 = C(parsers.line) * parsers.skipblanklines
6606 * Ct((defstart
6607 * parsers.indented_blocks(dlchunk)
6608 / self.parser_functions.parse_blocks_nested)^1)
6609 * Cc(false) / definition_list_item
6610
6611 local DefinitionListItemTight
6612 = C(parsers.line)
6613 * Ct((defstart * dlchunk
6614 / self.parser_functions.parse_blocks_nested)^1)
6615 * Cc(true) / definition_list_item
6616
6617 local DefinitionList
6618 = (Ct(DefinitionListItemLoose^1) * Cc(false)
6619 + Ct(DefinitionListItemTight^1)
6620 * (parsers.skipblanklines
6621 * -DefinitionListItemLoose * Cc(true))

```

```

6622) / writer.definitionlist
6623
6624 self.insert_pattern("Block after Heading",
6625 DefinitionList, "DefinitionList")
6626 end
6627 }
6628 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.

```

6629 M.extensions.fenced_code = function(blank_before_code_fence)
6630 return {
6631 name = "built-in fenced_code syntax extension",
6632 extend_writer = function(self)
6633 local options = self.options
6634

```

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

```

6635 function self.fencedCode(s, i)
6636 if not self.is_writing then return "" end
6637 local name = util.cache_verbatim(options.cacheDir, s)
6638 return {"\\markdownRendererInputFencedCode{" ,
6639 name,"}{" ,self.string(i),"}"}
6640 end
6641 end, extend_reader = function(self)
6642 local parsers = self.parsers
6643 local writer = self.writer
6644
6645 local FencedCode = (parsers.TildeFencedCode
6646 + parsers.BacktickFencedCode)
6647 / function(infostring, code)
6648 local expanded_code = self.expandtabs(code)
6649 return writer.fencedCode(expanded_code,
6650 infostring)
6651 end
6652
6653 self.insert_pattern("Block after Verbatim",
6654 FencedCode, "FencedCode")
6655
6656 local fencestart
6657 if blank_before_code_fence then
6658 fencestart = parsers.fail
6659 else

```

```

6660 fencestart = parsers.fencehead(parsers.backtick)
6661 + parsers.fencehead(parsers.tilde)
6662 end
6663
6664 local EndlineExceptions = parsers.EndlineExceptions + fencestart
6665 self.update_rule("EndlineExceptions", EndlineExceptions)
6666
6667 self.add_special_character("~")
6668 end
6669 }
6670 end

```

**3.1.6.5 Notes** The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

6671 M.extensions.notes = function(notes, inline_notes)
6672 assert(notes or inline_notes)
6673 return {
6674 name = "built-in notes syntax extension",
6675 extend_writer = function(self)

```

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

```

6676 function self.note(s)
6677 return {"\\markdownRendererNote{",s,"}"}
6678 end
6679 end, extend_reader = function(self)
6680 local parsers = self.parsers
6681 local writer = self.writer
6682
6683 if inline_notes then
6684 local InlineNote
6685 = parsers.circumflex
6686 * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
6687 / writer.note
6688
6689 self.insert_pattern("Inline after Emph",
6690 InlineNote, "InlineNote")
6691 end
6692 if notes then
6693 local function strip_first_char(s)
6694 return s:sub(2)
6695 end
6696
6697 local RawNoteRef

```

```

6698 = #(parsers.lbracket * parsers.circumflex)
6699 * parsers.tag / strip_first_char
6700
6701 local rawnotes = {}
6702
6703 -- like indirect_link
6704 local function lookup_note(ref)
6705 return writer.defer_call(function()
6706 local found = rawnotes[self.normalize_tag(ref)]
6707 if found then
6708 return writer.note(
6709 self.parser_functions.parse_blocks_nested(found))
6710 else
6711 return {"[",
6712 self.parser_functions.parse_inlines("^" .. ref), "]" }
6713 end
6714 end)
6715 end
6716
6717 local function register_note(ref,rawnote)
6718 rawnotes[self.normalize_tag(ref)] = rawnote
6719 return ""
6720 end
6721
6722 local NoteRef = RawNoteRef / lookup_note
6723
6724 local NoteBlock
6725 = parsers.leader * RawNoteRef * parsers.colon
6726 * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6727 / register_note
6728
6729 local Blank = NoteBlock + parsers.Blank
6730 self.update_rule("Blank", Blank)
6731
6732 self.insert_pattern("Inline after Emph",
6733 NoteRef, "NoteRef")
6734 end
6735
6736 self.add_special_character("^")
6737 end
6738 }
6739 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.

```

6740 M.extensions.header_attributes = function()

```



```

6741 return {
6742 name = "built-in header_attributes syntax extension",
6743 extend_writer = function()
6744 end, extend_reader = function(self)
6745 local parsers = self.parsers
6746 local writer = self.writer
6747
6748 local AtxHeading = Cg(parsers.heading_start, "level")
6749 * parsers.optionalspace
6750 * (C(((parsers.linechar
6751 - ((parsers.hash^1
6752 * parsers.optionalspace
6753 * parsers.attributes^-1
6754 + parsers.attributes)
6755 * parsers.optionalspace
6756 * parsers.newline)))
6757 * (parsers.linechar
6758 - parsers.hash
6759 - parsers.lbrace)^0)^1)
6760 / self.parser_functions.parse_inlines)
6761 * Cg(Ct(parsers.newline
6762 + (parsers.hash^1
6763 * parsers.optionalspace
6764 * parsers.attributes^-1
6765 + parsers.attributes)
6766 * parsers.optionalspace
6767 * parsers.newline), "attributes")
6768 * Cb("level")
6769 * Cb("attributes")
6770 / writer.heading
6771
6772 local SetextHeading = #(parsers.line * S("--"))
6773 * (C(((parsers.linechar
6774 - (parsers.attributes
6775 * parsers.optionalspace
6776 * parsers.newline))
6777 * (parsers.linechar
6778 - parsers.lbrace)^0)^1)
6779 / self.parser_functions.parse_inlines)
6780 * Cg(Ct(parsers.newline
6781 + (parsers.attributes
6782 * parsers.optionalspace
6783 * parsers.newline)), "attributes")
6784 * parsers.heading_level
6785 * Cb("attributes")
6786 * parsers.optionalspace
6787 * parsers.newline

```

```

6788 / writer.heading
6789
6790 local Heading = AtxHeading + SetextHeading
6791 self.update_rule("Heading", Heading)
6792 end
6793 }
6794 end

```

**3.1.6.7 YAML Metadata** The `extensions.jekyll_data` function implements the Pandoc `yaml_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

```

6795 M.extensions.jekyll_data = function(expect_jekyll_data)
6796 return {
6797 name = "built-in jekyll_data syntax extension",
6798 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`.

```

6799 function self.jekyllData(d, t, p)
6800 if not self.is_writing then return "" end
6801
6802 local buf = {}
6803
6804 local keys = {}
6805 for k, _ in pairs(d) do
6806 table.insert(keys, k)
6807 end
6808 table.sort(keys)
6809
6810 if not p then
6811 table.insert(buf, "\\markdownRendererJekyllDataBegin")
6812 end
6813
6814 if #d > 0 then
6815 table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
6816 table.insert(buf, self.uri(p or "null"))
6817 table.insert(buf, "}{")
6818 table.insert(buf, #keys)
6819 table.insert(buf, "}")
6820 else
6821 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
6822 table.insert(buf, self.uri(p or "null"))

```

```

6823 table.insert(buf, "}{")
6824 table.insert(buf, #keys)
6825 table.insert(buf, "}")
6826 end
6827
6828 for _, k in ipairs(keys) do
6829 local v = d[k]
6830 local typ = type(v)
6831 k = tostring(k or "null")
6832 if typ == "table" and next(v) ~= nil then
6833 table.insert(
6834 buf,
6835 self.jekyllData(v, t, k)
6836)
6837 else
6838 k = self.uri(k)
6839 v = tostring(v)
6840 if typ == "boolean" then
6841 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6842 table.insert(buf, k)
6843 table.insert(buf, "}{")
6844 table.insert(buf, v)
6845 table.insert(buf, "}")
6846 elseif typ == "number" then
6847 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6848 table.insert(buf, k)
6849 table.insert(buf, "}{")
6850 table.insert(buf, v)
6851 table.insert(buf, "}")
6852 elseif typ == "string" then
6853 table.insert(buf, "\\markdownRendererJekyllDataString{")
6854 table.insert(buf, k)
6855 table.insert(buf, "}{")
6856 table.insert(buf, t(v))
6857 table.insert(buf, "}")
6858 elseif typ == "table" then
6859 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
6860 table.insert(buf, k)
6861 table.insert(buf, "}")
6862 else
6863 error(format("Unexpected type %s for value of " ..
6864 "YAML key %s", typ, k))
6865 end
6866 end
6867 end
6868
6869 if #d > 0 then

```

```

6870 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6871 else
6872 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6873 end
6874
6875 if not p then
6876 table.insert(buf, "\\markdownRendererJekyllDataEnd")
6877 end
6878
6879 return buf
6880 end
6881 end, extend_reader = function(self)
6882 local parsers = self.parsers
6883 local writer = self.writer
6884
6885 local JekyllData
6886 = Cmt(C((parsers.line - P("---") - P("..."))^0)
6887 , function(s, i, text) -- luacheck: ignore s i
6888 local data
6889 local ran_ok, _ = pcall(function()
6890 local tinyyaml = require("markdown-tinyyaml")
6891 data = tinyyaml.parse(text, {timestamps=false})
6892 end)
6893 if ran_ok and data ~= nil then
6894 return true, writer.jekyllData(data, function(s)
6895 return self.parser_functions.parse_blocks_nested(s)
6896 end, nil)
6897 else
6898 return false
6899 end
6900 end
6901)
6902
6903 local UnexpectedJekyllData
6904 = P("---")
6905 * parsers.blankline / 0
6906 * #(-parsers.blankline) -- if followed by blank, it's thematic break
6907 * JekyllData
6908 * (P("---") + P("..."))
6909
6910 local ExpectedJekyllData
6911 = (P("---")
6912 * parsers.blankline / 0
6913 * #(-parsers.blankline) -- if followed by blank, it's thematic break
6914)^-1
6915 * JekyllData
6916 * (P("---") + P("..."))^-1

```

```

6917
6918 self.insert_pattern("Block before Blockquote",
6919 UnexpectedJekyllData, "UnexpectedJekyllData")
6920 if expect_jekyll_data then
6921 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
6922 end
6923 end
6924 }
6925 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.

```

6926 M.extensions.pipe_tables = function(table_captions)
6927
6928 local function make_pipe_table_rectangular(rows)
6929 local num_columns = #rows[2]
6930 local rectangular_rows = {}
6931 for i = 1, #rows do
6932 local row = rows[i]
6933 local rectangular_row = {}
6934 for j = 1, num_columns do
6935 rectangular_row[j] = row[j] or ""
6936 end
6937 table.insert(rectangular_rows, rectangular_row)
6938 end
6939 return rectangular_rows
6940 end
6941
6942 local function pipe_table_row(allow_empty_first_column
6943 , nonempty_column
6944 , column_separator
6945 , column)
6946 local row_beginning
6947 if allow_empty_first_column then
6948 row_beginning = -- empty first column
6949 #(parsers.spacechar^4
6950 * column_separator)
6951 * parsers.optionalspace
6952 * column
6953 * parsers.optionalspace
6954 -- non-empty first column
6955 + parsers.nonindentSPACE
6956 * nonempty_column^-1
6957 * parsers.optionalspace

```

```

6958 else
6959 row_beginning = parsers.nonindentSPACE
6960 * nonempty_column^-1
6961 * parsers.optionalspace
6962 end
6963
6964 return Ct(row_beginning
6965 * (-- single column with no leading pipes
6966 #(column_separator
6967 * parsers.optionalspace
6968 * parsers.newline)
6969 * column_separator
6970 * parsers.optionalspace
6971 -- single column with leading pipes or
6972 -- more than a single column
6973 + (column_separator
6974 * parsers.optionalspace
6975 * column
6976 * parsers.optionalspace)^1
6977 * (column_separator
6978 * parsers.optionalspace)^-1))
6979 end
6980
6981 return {
6982 name = "built-in pipe_tables syntax extension",
6983 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.

```

6984 function self.table(rows, caption)
6985 if not self.is_writing then return "" end
6986 local buffer = {"\markdownRendererTable{",
6987 caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
6988 local temp = rows[2] -- put alignments on the first row
6989 rows[2] = rows[1]
6990 rows[1] = temp
6991 for i, row in ipairs(rows) do
6992 table.insert(buffer, "{")
6993 for _, column in ipairs(row) do
6994 if i > 1 then -- do not use braces for alignments
6995 table.insert(buffer, "{")
6996 end
6997 table.insert(buffer, column)
6998 if i > 1 then
6999 table.insert(buffer, "}")
7000 end

```

```

7001 end
7002 table.insert(buffer, "}")
7003 end
7004 return buffer
7005 end
7006 end, extend_reader = function(self)
7007 local parsers = self.parsers
7008 local writer = self.writer
7009
7010 local table_hline_separator = parsers.pipe + parsers.plus
7011
7012 local table_hline_column = (parsers.dash
7013 - #(parsers.dash
7014 * (parsers.spacechar
7015 + table_hline_separator
7016 + parsers.newline)))^1
7017 * (parsers.colon * Cc("r")
7018 + parsers.dash * Cc("d"))
7019 + parsers.colon
7020 * (parsers.dash
7021 - #(parsers.dash
7022 * (parsers.spacechar
7023 + table_hline_separator
7024 + parsers.newline)))^1
7025 * (parsers.colon * Cc("c")
7026 + parsers.dash * Cc("l"))
7027
7028 local table_hline = pipe_table_row(false
7029 , table_hline_column
7030 , table_hline_separator
7031 , table_hline_column)
7032
7033 local table_caption_beginning = parsers.skipblanklines
7034 * parsers.nonindentspace
7035 * (P("Table")^-1 * parsers.colon)
7036 * parsers.optionalspace
7037
7038 local table_row = pipe_table_row(true
7039 , (C((parsers.linechar - parsers.pipe)^1)
7040 / self.parser_functions.parse_inlines)
7041 , parsers.pipe
7042 , (C((parsers.linechar - parsers.pipe)^0)
7043 / self.parser_functions.parse_inlines))
7044
7045 local table_caption
7046 if table_captions then
7047 table_caption = #table_caption_beginning

```

```

7048 * table_caption_beginning
7049 * Ct(parsers.IndentedInline^1)
7050 * parsers.newline
7051 else
7052 table_caption = parsers.fail
7053 end
7054
7055 local PipeTable = Ct(table_row * parsers.newline
7056 * table_hline
7057 * (parsers.newline * table_row)^0)
7058 / make_pipe_table_rectangular
7059 * table_caption^-1
7060 / writer.table
7061
7062 self.insert_pattern("Block after Blockquote",
7063 PipeTable, "PipeTable")
7064 end
7065 }
7066 end

```

**3.1.6.9 Raw Attributes** The `extensions.raw_attribute` function implements the Pandoc raw attribute syntax extension.

```

7067 M.extensions.raw_attribute = function()
7068 return {
7069 name = "built-in raw_attribute syntax extension",
7070 extend_writer = function(self)
7071 local options = self.options
7072

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `i` to the output format.

```

7073 function self.rawInline(s, attr)
7074 if not self.is_writing then return "" end
7075 local name = util.cache_verbatim(options.cacheDir, s)
7076 return {"\\markdownRendererInputRawInline{" ,
7077 name,"}{" , self.string(attr),"}"}
7078 end
7079
7080 if options.fencedCode then

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `i` to the output format.

```

7081 function self.rawBlock(s, attr)
7082 if not self.is_writing then return "" end
7083 local name = util.cache_verbatim(options.cacheDir, s)
7084 return {"\\markdownRendererInputRawBlock{" ,
7085 name,"}{" , self.string(attr),"}"}

```



```

7086 end
7087 end
7088 end, extend_reader = function(self)
7089 local options = self.options
7090 local writer = self.writer
7091
7092 local raw_attribute = parsers.lbrace
7093 * parsers.optionalspace
7094 * parsers.equal
7095 * C(parsers.attribute_key)
7096 * parsers.optionalspace
7097 * parsers.rbrace
7098
7099 local RawInline = parsers.inticks
7100 * raw_attribute
7101 / writer.rawInline
7102
7103 self.insert_pattern("Inline before Code",
7104 RawInline, "RawInline")
7105
7106 if options.fencedCode then
7107 local RawBlock = (parsers.TildeFencedCode
7108 + parsers.BacktickFencedCode)
7109 / function(infostring, code)
7110 local expanded_code = self.expandtabs(code)
7111 local attr = lpeg.match(raw_attribute, infostring)
7112 if attr then
7113 return writer.rawBlock(expanded_code, attr)
7114 else
7115 return writer.fencedCode(expanded_code,
7116 infostring)
7117 end
7118 end
7119
7120 self.insert_pattern("Block after Verbatim",
7121 RawBlock, "RawBlock")
7122 end
7123 end
7124 }
7125 end

```

**3.1.6.10 Strike-Through** The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

7126 M.extensions.strike_through = function()
7127 return {
7128 name = "built-in strike_through syntax extension",

```

```
7129 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.

```
7130 function self.strike_through(s)
7131 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
7132 end
7133 end, extend_reader = function(self)
7134 local parsers = self.parsers
7135 local writer = self.writer
7136
7137 local StrikeThrough = (
7138 parsers.between(parsers.Inline, parsers.doubletildes,
7139 parsers.doubletildes)
7140) / writer.strike_through
7141
7142 self.insert_pattern("Inline after Emph",
7143 StrikeThrough, "StrikeThrough")
7144
7145 self.add_special_character("~")
7146 end
7147 }
7148 end
```

**3.1.6.11 Superscripts** The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```
7149 M.extensions.superscripts = function()
7150 return {
7151 name = "built-in superscripts syntax extension",
7152 extend_writer = function(self)
```

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

```
7153 function self.superscript(s)
7154 return {"\\markdownRendererSuperscript{" ,s,"}"}
7155 end
7156 end, extend_reader = function(self)
7157 local parsers = self.parsers
7158 local writer = self.writer
7159
7160 local Superscript = (
7161 parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
7162) / writer.superscript
7163
7164 self.insert_pattern("Inline after Emph",
7165 Superscript, "Superscript")
7166
```

```

7167 self.add_special_character("^")
7168 end
7169 }
7170 end

```

**3.1.6.12 Subscripts** The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

7171 M.extensions.subscripts = function()
7172 return {
7173 name = "built-in subscripts syntax extension",
7174 extend_writer = function(self)

```

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

```

7175 function self.subscript(s)
7176 return {"\\markdownRendererSubscript{" ,s,"}"}
7177 end
7178 end, extend_reader = function(self)
7179 local parsers = self.parsers
7180 local writer = self.writer
7181
7182 local Subscript = (
7183 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
7184) / writer.subscript
7185
7186 self.insert_pattern("Inline after Emph",
7187 Subscript, "Subscript")
7188
7189 self.add_special_character("~")
7190 end
7191 }
7192 end

```

**3.1.6.13 Fancy Lists** The `extensions.fancy_lists` function implements the Pandoc fancy list extension.

```

7193 M.extensions.fancy_lists = function()
7194 return {
7195 name = "built-in fancy_lists syntax extension",
7196 extend_writer = function(self)
7197 local options = self.options
7198

```

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.

```

7199 function self.fancylist(items,tight,startnum,numstyle,numdelim)
7200 if not self.is_writing then return "" end
7201 local buffer = {}
7202 local num = startnum
7203 for _,item in ipairs(items) do
7204 buffer[#buffer + 1] = self.fancyitem(item,num)
7205 if num ~= nil then
7206 num = num + 1
7207 end
7208 end
7209 local contents = util.intersperse(buffer,"\n")
7210 if tight and options.tightLists then
7211 return {"\\markdownRendererFancy01BeginTight{" ,
7212 numstyle,"}{",numdelim,"}",contents,
7213 "\\n\\markdownRendererFancy01EndTight "}
7214 else
7215 return {"\\markdownRendererFancy01Begin{" ,
7216 numstyle,"}{",numdelim,"}",contents,
7217 "\\n\\markdownRendererFancy01End "}
7218 end
7219 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.

```

7220 function self.fancyitem(s,num)
7221 if num ~= nil then
7222 return {"\\markdownRendererFancy01ItemWithNumber{" ,num,"}",s,
7223 "\\n\\markdownRendererFancy01ItemEnd "}
7224 else
7225 return {"\\markdownRendererFancy01Item " ,s,"\\n\\markdownRendererFancy01ItemEnd "
7226 end
7227 end

```

```

7228 end, extend_reader = function(self)
7229 local parsers = self.parsers
7230 local options = self.options
7231 local writer = self.writer
7232
7233 local label = parsers.dig + parsers.letter
7234 local numdelim = parsers.period + parsers.rparent
7235 local enumerator = C(label^3 * numdelim) * #parsers.spacing
7236 + C(label^2 * numdelim) * #parsers.spacing
7237 * (parsers.tab + parsers.space^1)
7238 + C(label * numdelim) * #parsers.spacing
7239 * (parsers.tab + parsers.space^-2)
7240 + parsers.space * C(label^2 * numdelim)
7241 * #parsers.spacing
7242 + parsers.space * C(label * numdelim)
7243 * #parsers.spacing
7244 * (parsers.tab + parsers.space^-1)
7245 + parsers.space * parsers.space * C(label^1
7246 * numdelim) * #parsers.spacing
7247 local starter = parsers.bullet + enumerator
7248
7249 local NestedList = Cs((parsers.optionallyindentedline
7250 - starter)^1)
7251 / function(a) return "\001"..a end
7252
7253 local ListBlockLine = parsers.optionallyindentedline
7254 - parsers.blankline - (parsers.indent^-1
7255 * starter)
7256
7257 local ListBlock = parsers.line * ListBlockLine^0
7258
7259 local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
7260 * ListBlock
7261
7262 local TightListItem = function(starter)
7263 return -parsers.ThematicBreak
7264 * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
7265 / self.parser_functions.parse_blocks_nested)
7266 * -(parsers.blanklines * parsers.indent)
7267 end
7268
7269 local LooseListItem = function(starter)
7270 return -parsers.ThematicBreak
7271 * Cs(starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
7272 * (NestedList + ListContinuationBlock^0)
7273 * (parsers.blanklines / "\n\n"))

```

```

7274) / self.parser_functions.parse_blocks_nested
7275 end
7276
7277 local function roman2number(roman)
7278 local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
7279 local numeral = 0
7280
7281 local i = 1
7282 local len = string.len(roman)
7283 while i < len do
7284 local z1, z2 = romans[string.sub(roman, i, i)], romans[string.sub(roman, i+1, i+1)]
7285 if z1 < z2 then
7286 numeral = numeral + (z2 - z1)
7287 i = i + 2
7288 else
7289 numeral = numeral + z1
7290 i = i + 1
7291 end
7292 end
7293 if i <= len then numeral = numeral + romans[string.sub(roman,i,i)] end
7294 return numeral
7295 end
7296
7297 local function sniffstyle(itemprefix)
7298 local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])")
7299 local numdelim
7300 if delimend == ")" then
7301 numdelim = "OneParen"
7302 elseif delimend == "." then
7303 numdelim = "Period"
7304 else
7305 numdelim = "Default"
7306 end
7307 numstr = numstr or itemprefix
7308
7309 local num
7310 num = numstr:match("^([IVXL]+)")
7311 if num then
7312 return roman2number(num), "UpperRoman", numdelim
7313 end
7314 num = numstr:match("^([ivxl]+)")
7315 if num then
7316 return roman2number(string.upper(num)), "LowerRoman", numdelim
7317 end
7318 num = numstr:match("^([A-Z])")
7319 if num then
7320 return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim

```

```

7321 end
7322 num = numstr:match("^([a-z])")
7323 if num then
7324 return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7325 end
7326 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7327 end
7328
7329 local function fancylist(items,tight,start)
7330 local startnum, numstyle, numdelim = sniffstyle(start)
7331 return writer.fancylist(items,tight,
7332 options.startNumber and startnum,
7333 numstyle or "Decimal",
7334 numdelim or "Default")
7335 end
7336
7337 local FancyList = Cg(enumerator, "listtype") *
7338 (Ct(TightListItem(Cb("listtype"))
7339 * TightListItem(enumerator)^0
7340 * Cc(true) * parsers.skipblanklines * -enumerator
7341 + Ct(LooseListItem(Cb("listtype"))
7342 * LooseListItem(enumerator)^0
7343 * Cc(false) * parsers.skipblanklines
7344) * Cb("listtype") / fancylist
7345
7346 self.update_rule("OrderedList", FancyList)
7347 end
7348 }
7349 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 2.1.1.

```

7350 function M.new(options)

```

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

```

7351 options = options or {}
7352 setmetatable(options, { __index = function (_, key)
7353 return defaultOptions[key] end })

```

Apply built-in syntax extensions based on `options`.

```

7354 local extensions = {}
7355
7356 if options.contentBlocks then
7357 local content_blocks_extension = M.extensions.content_blocks(
7358 options.contentBlocksLanguageMap)

```

```

7359 table.insert(extensions, content_blocks_extension)
7360 end
7361
7362 if options.definitionLists then
7363 local definition_lists_extension = M.extensions.definition_lists(
7364 options.tightLists)
7365 table.insert(extensions, definition_lists_extension)
7366 end
7367
7368 if options.fencedCode then
7369 local fenced_code_extension = M.extensions.fenced_code(
7370 options.blankBeforeCodeFence)
7371 table.insert(extensions, fenced_code_extension)
7372 end
7373
7374 if options.headerAttributes then
7375 local header_attributes_extension = M.extensions.header_attributes()
7376 table.insert(extensions, header_attributes_extension)
7377 end
7378
7379 if options.jekyllData then
7380 local jekyll_data_extension = M.extensions.jekyll_data(
7381 options.expectJekyllData)
7382 table.insert(extensions, jekyll_data_extension)
7383 end
7384
7385 if options.pipeTables then
7386 local pipe_tables_extension = M.extensions.pipe_tables(
7387 options.tableCaptions)
7388 table.insert(extensions, pipe_tables_extension)
7389 end
7390
7391 if options.rawAttribute then
7392 local raw_attribute_extension = M.extensions.raw_attribute()
7393 table.insert(extensions, raw_attribute_extension)
7394 end
7395
7396 if options.strikeThrough then
7397 local strike_through_extension = M.extensions.strike_through()
7398 table.insert(extensions, strike_through_extension)
7399 end
7400
7401 if options.subscripts then
7402 local subscript_extension = M.extensions.subscripts()
7403 table.insert(extensions, subscript_extension)
7404 end
7405

```



```

7406 if options.superscripts then
7407 local superscript_extension = M.extensions.superscripts()
7408 table.insert(extensions, superscript_extension)
7409 end
7410

```

The footnotes and inlineFootnotes option has been deprecated and will be removed in Markdown 3.0.0.

```

7411 if options.footnotes or options.inlineFootnotes or
7412 options.notes or options.inlineNotes then
7413 local notes_extension = M.extensions.notes(
7414 options.footnotes or options.notes,
7415 options.inlineFootnotes or options.inlineNotes)
7416 table.insert(extensions, notes_extension)
7417 end
7418
7419 if options.citations then
7420 local citations_extension = M.extensions.citations(options.citationNbsps)
7421 table.insert(extensions, citations_extension)
7422 end
7423
7424 if options.fancyLists then
7425 local fancy_lists_extension = M.extensions.fancy_lists()
7426 table.insert(extensions, fancy_lists_extension)
7427 end

```

Apply user-defined syntax extensions based on [options.extensions](#).

```

7428 for _, user_extension_filename in ipairs(options.extensions) do
7429 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

7430 local pathname = util.lookup_files(filename)
7431 local input_file = assert(io.open(pathname, "r"),
7432 [[Could not open user-defined syntax extension "]]
7433 .. pathname .. [[for reading]])
7434 local input = assert(input_file:read("*a"))
7435 assert(input_file:close())
7436 local user_extension, err = load([[
7437 local sandbox = {}
7438 setmetatable(sandbox, {__index = _G})
7439 _ENV = sandbox
7440]] .. input)()
7441 assert(user_extension,
7442 [[Failed to compile user-defined syntax extension "]]
7443 .. pathname .. [[:]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

7444 assert(user_extension.api_version ~= nil,
7445 [[User-defined syntax extension "]] .. pathname

```

```

7446 .. [{" does not specify mandatory field "api_version"}])
7447 assert(type(user_extension.api_version) == "number",
7448 [{"User-defined syntax extension "} .. pathname
7449 .. [{" specifies field "api_version" of type "}]]
7450 .. type(user_extension.api_version)
7451 .. [{" but "number" was expected}])
7452 assert(user_extension.api_version > 0
7453 and user_extension.api_version <= metadata.user_extension_api_version,
7454 [{"User-defined syntax extension "} .. pathname
7455 .. [{" uses syntax extension API version "}]]
7456 .. user_extension.api_version .. [{" but markdown.lua }]
7457 .. metadata.version .. [{" uses API version }]
7458 .. metadata.user_extension_api_version
7459 .. [{" , which is incompatible}])
7460
7461 assert(user_extension.grammar_version ~= nil,
7462 [{"User-defined syntax extension "} .. pathname
7463 .. [{" does not specify mandatory field "grammar_version"}])
7464 assert(type(user_extension.grammar_version) == "number",
7465 [{"User-defined syntax extension "} .. pathname
7466 .. [{" specifies field "grammar_version" of type "}]]
7467 .. type(user_extension.grammar_version)
7468 .. [{" but "number" was expected}])
7469 assert(user_extension.grammar_version == metadata.grammar_version,
7470 [{"User-defined syntax extension "} .. pathname
7471 .. [{" uses grammar version "}]] .. user_extension.grammar_version
7472 .. [{" but markdown.lua }] .. metadata.version
7473 .. [{" uses grammar version }] .. metadata.grammar_version
7474 .. [{" , which is incompatible}])
7475
7476 assert(user_extension.finalize_grammar ~= nil,
7477 [{"User-defined syntax extension "} .. pathname
7478 .. [{" does not specify mandatory "finalize_grammar" field}])
7479 assert(type(user_extension.finalize_grammar) == "function",
7480 [{"User-defined syntax extension "} .. pathname
7481 .. [{" specifies field "finalize_grammar" of type "}]]
7482 .. type(user_extension.finalize_grammar)
7483 .. [{" 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 3.1.6.)

```

7484 local extension = {
7485 name = [{"user-defined "} .. pathname .. [{" syntax extension}],
7486 extend_reader = user_extension.finalize_grammar,
7487 extend_writer = function() end,
7488 }
7489 return extension

```

```

7490 end)(user_extension_filename)
7491 table.insert(extensions, user_extension)
7492 end

```

Produce and return a conversion function from markdown to plain TeX.

```

7493 local writer = M.writer.new(options)
7494 local reader = M.reader.new(writer, options)
7495 local convert = reader.finalize_grammar(extensions)
7496
7497 return convert
7498 end
7499
7500 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.

```

7501
7502 local input
7503 if input_filename then
7504 local input_file = assert(io.open(input_filename, "r"),
7505 [[Could not open file]] .. input_filename .. [[for reading]])
7506 input = assert(input_file:read("*a"))
7507 assert(input_file:close())
7508 else
7509 input = assert(io.read("*a"))
7510 end
7511

```

First, ensure that the `options.cacheDir` directory exists.

```

7512 local lfs = require("lfs")
7513 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7514 assert(lfs.mkdir(options["cacheDir"]))
7515 end
7516
7517 local ran_ok, kpse = pcall(require, "kpse")
7518 if ran_ok then kpse.set_program_name("luatex") end
7519 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.

```

7520 if metadata.version ~= md.metadata.version then
7521 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7522 "markdown.lua " .. md.metadata.version .. ".")
7523 end
7524 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.

```

7525 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7526
7527 if output_filename then
7528 local output_file = assert(io.open(output_filename, "w"),
7529 [[Could not open file]] .. output_filename .. [{" for writing}])
7530 assert(output_file:write(output))
7531 assert(output_file:close())
7532 else
7533 assert(io.write(output))
7534 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

```

7535 \ifx\markdownInfo\undefined
7536 \def\markdownInfo#1{%
7537 \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1}}%
7538 \fi
7539 \ifx\markdownWarning\undefined
7540 \def\markdownWarning#1{%
7541 \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
7542 \fi
7543 \ifx\markdownError\undefined
7544 \def\markdownError#1#2{%
7545 \errhelp{#2.}%
7546 \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
7547 \fi

```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

7548 \def\markdownRendererInterblockSeparatorPrototype{\par}%
7549 \def\markdownRendererLineBreakPrototype{\hfil\break}%
7550 \let\markdownRendererEllipsisPrototype\dots
7551 \def\markdownRendererNbspPrototype{~}%
7552 \def\markdownRendererLeftBracePrototype{\char`{}%
7553 \def\markdownRendererRightBracePrototype{\char`\}}%
7554 \def\markdownRendererDollarSignPrototype{\char`$}%
7555 \def\markdownRendererPercentSignPrototype{\char`\}%

```

```

7556 \def\markdownRendererAmpersandPrototype{\&}%
7557 \def\markdownRendererUnderscorePrototype{\char`_}%
7558 \def\markdownRendererHashPrototype{\char`#}%
7559 \def\markdownRendererCircumflexPrototype{\char`^}%
7560 \def\markdownRendererBackslashPrototype{\char`\}%
7561 \def\markdownRendererTildePrototype{\char`~}%
7562 \def\markdownRendererPipePrototype{|}%
7563 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
7564 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
7565 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7566 \markdownInput{#3}}%
7567 \def\markdownRendererContentBlockOnlineImagePrototype{%
7568 \markdownRendererImage}%
7569 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
7570 \markdownRendererInputFencedCode{#3}{#2}}%
7571 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
7572 \def\markdownRendererUlBeginPrototype{}%
7573 \def\markdownRendererUlBeginTightPrototype{}%
7574 \def\markdownRendererUlItemPrototype{}%
7575 \def\markdownRendererUlItemEndPrototype{}%
7576 \def\markdownRendererUlEndPrototype{}%
7577 \def\markdownRendererUlEndTightPrototype{}%
7578 \def\markdownRendererOlBeginPrototype{}%
7579 \def\markdownRendererOlBeginTightPrototype{}%
7580 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
7581 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
7582 \def\markdownRendererOlItemPrototype{}%
7583 \def\markdownRendererOlItemWithNumberPrototype#1{}%
7584 \def\markdownRendererOlItemEndPrototype{}%
7585 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
7586 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
7587 \def\markdownRendererFancyOlItemEndPrototype{}%
7588 \def\markdownRendererOlEndPrototype{}%
7589 \def\markdownRendererOlEndTightPrototype{}%
7590 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
7591 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
7592 \def\markdownRendererDlBeginPrototype{}%
7593 \def\markdownRendererDlBeginTightPrototype{}%
7594 \def\markdownRendererDlItemPrototype#1{#1}%
7595 \def\markdownRendererDlItemEndPrototype{}%
7596 \def\markdownRendererDlDefinitionBeginPrototype{}%
7597 \def\markdownRendererDlDefinitionEndPrototype{\par}%
7598 \def\markdownRendererDlEndPrototype{}%
7599 \def\markdownRendererDlEndTightPrototype{}%
7600 \def\markdownRendererEmphasisPrototype#1{\it#1}%
7601 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
7602 \def\markdownRendererBlockQuoteBeginPrototype{\par\begin\group\it}%

```

```

7603 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
7604 \def\markdownRendererInputVerbatimPrototype#1{%
7605 \par{\tt\input#1\relax{}}\par}%
7606 \def\markdownRendererInputFencedCodePrototype#1#2{%
7607 \markdownRendererInputVerbatimPrototype{#1}}%
7608 \def\markdownRendererHeadingOnePrototype#1{#1}%
7609 \def\markdownRendererHeadingTwoPrototype#1{#1}%
7610 \def\markdownRendererHeadingThreePrototype#1{#1}%
7611 \def\markdownRendererHeadingFourPrototype#1{#1}%
7612 \def\markdownRendererHeadingFivePrototype#1{#1}%
7613 \def\markdownRendererHeadingSixPrototype#1{#1}%
7614 \def\markdownRendererThematicBreakPrototype{}%
7615 \def\markdownRendererNotePrototype#1{#1}%
7616 \def\markdownRendererCitePrototype#1{}%
7617 \def\markdownRendererTextCitePrototype#1{}%
7618 \def\markdownRendererTickedBoxPrototype{[X]}%
7619 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
7620 \def\markdownRendererUntickedBoxPrototype{[]}%
7621 \def\markdownRendererStrikeThroughPrototype#1{#1}%
7622 \def\markdownRendererSuperscriptPrototype#1{#1}%
7623 \def\markdownRendererSubscriptPrototype#1{#1}%

```

**3.2.2.1 Raw Attribute Renderer Prototypes** In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

7624 \ExplSyntaxOn
7625 \cs_gset:Npn
7626 \markdownRendererInputRawInlinePrototype#1#2
7627 {
7628 \str_case:nn
7629 { #2 }
7630 {
7631 { tex } { \markdownEscape{#1} }
7632 { md } { \markdownInput{#1} }
7633 }
7634 }
7635 \cs_gset_eq:NN
7636 \markdownRendererInputRawBlockPrototype
7637 \markdownRendererInputRawInlinePrototype
7638 \ExplSyntaxOff

```

**3.2.2.2 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_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```
7639 \ExplSyntaxOn
7640 \seq_new:N \g_@@_jekyll_data_datatypes_seq
7641 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
7642 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
7643 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { 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.

```
7644 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
7645 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
7646 {
7647 \seq_if_empty:NF
7648 \g_@@_jekyll_data_datatypes_seq
7649 {
7650 \seq_get_right:NN
7651 \g_@@_jekyll_data_datatypes_seq
7652 \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.

```
7653 \str_if_eq:NNTF
7654 \l_tmpa_tl
7655 \c_@@_jekyll_data_sequence_tl
7656 {
7657 \seq_put_right:Nn
7658 \g_@@_jekyll_data_wildcard_absolute_address_seq
7659 { * }
7660 }
7661 {
7662 \seq_put_right:Nn
7663 \g_@@_jekyll_data_wildcard_absolute_address_seq
```

```

7664 { #1 }
7665 }
7666 }
7667 }

```

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.

```

7668 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
7669 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
7670 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
7671 {
7672 \seq_pop_left:NN #1 \l_tmpa_tl
7673 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
7674 \seq_put_left:NV #1 \l_tmpa_tl
7675 }
7676 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
7677 {
7678 \markdown_jekyll_data_concatenate_address:NN
7679 \g_@@_jekyll_data_wildcard_absolute_address_seq
7680 \g_@@_jekyll_data_wildcard_absolute_address_tl
7681 \seq_get_right:NN
7682 \g_@@_jekyll_data_wildcard_absolute_address_seq

```



```

7683 \g_@@_jekyll_data_wildcard_relative_address_tl
7684 }

```

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.

```

7685 \cs_new:Nn \markdown_jekyll_data_push:nN
7686 {
7687 \markdown_jekyll_data_push_address_segment:n
7688 { #1 }
7689 \seq_put_right:NV
7690 \g_@@_jekyll_data_datatypes_seq
7691 #2
7692 \markdown_jekyll_data_update_address_tls:
7693 }
7694 \cs_new:Nn \markdown_jekyll_data_pop:
7695 {
7696 \seq_pop_right:NN
7697 \g_@@_jekyll_data_wildcard_absolute_address_seq
7698 \l_tmpa_tl
7699 \seq_pop_right:NN
7700 \g_@@_jekyll_data_datatypes_seq
7701 \l_tmpa_tl
7702 \markdown_jekyll_data_update_address_tls:
7703 }

```

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.

```

7704 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
7705 {
7706 \keys_set_known:nn
7707 { markdown/jekyllData }
7708 { { #1 } = { #2 } }
7709 }
7710 \cs_generate_variant:Nn
7711 \markdown_jekyll_data_set_keyval:nn
7712 { Vn }
7713 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
7714 {
7715 \markdown_jekyll_data_push:nN
7716 { #1 }
7717 \c_@@_jekyll_data_scalar_tl
7718 \markdown_jekyll_data_set_keyval:Vn
7719 \g_@@_jekyll_data_wildcard_absolute_address_tl
7720 { #2 }
7721 \markdown_jekyll_data_set_keyval:Vn
7722 \g_@@_jekyll_data_wildcard_relative_address_tl
7723 { #2 }

```

```

7724 \markdown_jekyll_data_pop:
7725 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

7726 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
7727 \markdown_jekyll_data_push:nN
7728 { #1 }
7729 \c_@@_jekyll_data_sequence_tl
7730 }
7731 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
7732 \markdown_jekyll_data_push:nN
7733 { #1 }
7734 \c_@@_jekyll_data_mapping_tl
7735 }
7736 \def\markdownRendererJekyllDataSequenceEndPrototype{
7737 \markdown_jekyll_data_pop:
7738 }
7739 \def\markdownRendererJekyllDataMappingEndPrototype{
7740 \markdown_jekyll_data_pop:
7741 }
7742 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
7743 \markdown_jekyll_data_set_keyvals:nn
7744 { #1 }
7745 { #2 }
7746 }
7747 \def\markdownRendererJekyllDataEmptyPrototype#1{}
7748 \def\markdownRendererJekyllDataNumberPrototype#1#2{
7749 \markdown_jekyll_data_set_keyvals:nn
7750 { #1 }
7751 { #2 }
7752 }
7753 \def\markdownRendererJekyllDataStringPrototype#1#2{
7754 \markdown_jekyll_data_set_keyvals:nn
7755 { #1 }
7756 { #2 }
7757 }
7758 \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 TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

7759 \ExplSyntaxOn
7760 \tl_new:N \g_@@_formatted_lua_options_tl
7761 \cs_new:Nn \@@_format_lua_options:

```

```

7762 {
7763 \tl_gclear:N
7764 \g_@@_formatted_lua_options_tl
7765 \seq_map_function:NN
7766 \g_@@_lua_options_seq
7767 \@@_format_lua_option:n
7768 }
7769 \cs_new:Nn \@@_format_lua_option:n
7770 {
7771 \@@_typecheck_option:n
7772 { #1 }
7773 \@@_get_option_type:nN
7774 { #1 }
7775 \l_tmpa_tl
7776 \bool_case_true:nF
7777 {
7778 {
7779 \str_if_eq_p:VV
7780 \l_tmpa_tl
7781 \c_@@_option_type_boolean_tl ||
7782 \str_if_eq_p:VV
7783 \l_tmpa_tl
7784 \c_@@_option_type_number_tl ||
7785 \str_if_eq_p:VV
7786 \l_tmpa_tl
7787 \c_@@_option_type_counter_tl
7788 }
7789 {
7790 \@@_get_option_value:nN
7791 { #1 }
7792 \l_tmpa_tl
7793 \tl_gput_right:Nx
7794 \g_@@_formatted_lua_options_tl
7795 { #1~::~ \l_tmpa_tl ,~ }
7796 }
7797 }
7798 \str_if_eq_p:VV
7799 \l_tmpa_tl
7800 \c_@@_option_type_clist_tl
7801 }
7802 {
7803 \@@_get_option_value:nN
7804 { #1 }
7805 \l_tmpa_tl
7806 \tl_gput_right:Nx
7807 \g_@@_formatted_lua_options_tl
7808 { #1~::~\c_left_brace_str }

```

```

7809 \clist_map_inline:Vn
7810 \l_tmpa_tl
7811 {
7812 \tl_gput_right:Nx
7813 \g_@@_formatted_lua_options_tl
7814 { "##1" ,~ }
7815 }
7816 \tl_gput_right:Nx
7817 \g_@@_formatted_lua_options_tl
7818 { \c_right_brace_str ,~ }
7819 }
7820 }
7821 {
7822 \@@_get_option_value:nN
7823 { #1 }
7824 \l_tmpa_tl
7825 \tl_gput_right:Nx
7826 \g_@@_formatted_lua_options_tl
7827 { #1~::~ " \l_tmpa_tl " ,~ }
7828 }
7829 }
7830 \cs_generate_variant:Nn
7831 \clist_map_inline:nn
7832 { Vn }
7833 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
7834 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
7835 \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.

```
7836 \def\markdownPrepare{%
```

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

```

7837 local lfs = require("lfs")
7838 local cacheDir = "\markdownOptionCacheDir"
7839 if not lfs.isdir(cacheDir) then
7840 assert(lfs.mkdir(cacheDir))
7841 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.

```

7842 local md = require("markdown")
7843 local convert = md.new(\markdownLuaOptions)
7844 }%

```

### 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.

```
7845 \ExplSyntaxOn
7846 \cs_new:Nn
7847 \@@_if_option:nTF
7848 {
7849 \@@_get_option_type:nN
7850 { #1 }
7851 \l_tmpa_tl
7852 \str_if_eq:NNF
7853 \l_tmpa_tl
7854 \c_@@_option_type_boolean_tl
7855 {
7856 \msg_error:nxxx
7857 { @@ }
7858 { expected-boolean-option }
7859 { #1 }
7860 { \l_tmpa_tl }
7861 }
7862 \@@_get_option_value:nN
7863 { #1 }
7864 \l_tmpa_tl
7865 \str_if_eq:NNTF
7866 \l_tmpa_tl
7867 \c_@@_option_value_true_tl
7868 { #2 }
7869 { #3 }
7870 }
7871 \msg_new:nnn
7872 { @@ }
7873 { expected-boolean-option }
7874 {
7875 Option~#1~has~type~#2,~
7876 but~a~boolean~was~expected.
7877 }
7878 \let\markdownIfOption=\@@_if_option:nTF
7879 \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.

```
7880 \csname newread\endcsname\markdownInputFileStream
7881 \csname newwrite\endcsname\markdownOutputFileStream
```

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

```

7882 \begingroup
7883 \catcode`\^^I=12%
7884 \gdef\markdownReadAndConvertTab{^^I}%
7885 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX} 2_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

7886 \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.

```

7887 \catcode`\^^M=13%
7888 \catcode`\^^I=13%
7889 \catcode`|=0%
7890 \catcode`\=12%
7891 |catcode`@=14%
7892 |catcode`|=12@
7893 |gdef|markdownReadAndConvert#1#2{@
7894 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```

7895 |markdownIfOption{frozenCache}{-}{@
7896 |immediate|openout|markdownOutputFileStream@
7897 |markdownOptionInputTempFileName|relax@
7898 |markdownInfo{Buffering markdown input into the temporary @
7899 |input file "|markdownOptionInputTempFileName" and scanning @
7900 |for the closing token sequence "#1"}@
7901 }@

```

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

```

7902 |def|do##1{|catcode`##1=12}|dospecials@
7903 |catcode`|=12@
7904 |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 (`^^M`) are produced.

```

7905 |def|markdownReadAndConvertStripPercentSign##1{@
7906 |markdownIfOption{stripPercentSigns}{-}{@
7907 |if##1%@

```

```

7908 |expandafter|expandafter|expandafter@
7909 |markdownReadAndConvertProcessLine@
7910 |else@
7911 |expandafter|expandafter|expandafter@
7912 |markdownReadAndConvertProcessLine@
7913 |expandafter|expandafter|expandafter##1@
7914 |fi@
7915 }{@
7916 |expandafter@
7917 |markdownReadAndConvertProcessLine@
7918 |expandafter##1@
7919 }@
7920 }@

```

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 (`^^M`) are produced.

```

7921 |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.

```

7922 |ifx|relax##3|relax@
7923 |markdownIfOption{frozenCache}{-}{@
7924 |immediate|write|markdownOutputFileStream{##1}@
7925 }@
7926 |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.

```

7927 |def^^M{@
7928 |markdownInfo{The ending token sequence was found}@
7929 |markdownIfOption{frozenCache}{-}{@
7930 |immediate|closeout|markdownOutputFileStream@
7931 }@
7932 |endgroup@
7933 |markdownInput{@
7934 |markdownOptionOutputDir@
7935 |/\markdownOptionInputTempFileName@
7936 }@
7937 #2}@
7938 |fi@

```

Repeat with the next line.

```
7939 ^^M}@
```

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

```
7940 |catcode`|^I=13@
```

```
7941 |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.

```
7942 |catcode`|^M=13@
```

```
7943 |def^^M##1^^M{@
```

```
7944 |def^^M####1^^M{@
```

```
7945 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
```

```
7946 ^^M}@
```

```
7947 ^^M}@
```

Reset the character categories back to the former state.

```
7948 |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.

```
7949 \ExplSyntaxOn
```

```
7950 \int_compare:nT
```

```
7951 { \markdownMode = 3 }
```

```
7952 {
```

```
7953 \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
```

```
7954 \file_input:n { lt3luabridge.tex }
```

```
7955 \cs_new:Npn
```

```
7956 \markdownLuaExecute
```

```
7957 { \luabridgeExecute }
```

```
7958 }
```

```
7959 \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].



```

7960 \ifnum\markdownMode<2\relax
7961 \ifnum\markdownMode=0\relax
7962 \markdownWarning{Using mode 0: Shell escape via write18
7963 (deprecated, to be removed in Markdown 3.0.0)}%
7964 \else
7965 \markdownWarning{Using mode 1: Shell escape via os.execute
7966 (deprecated, to be removed in Markdown 3.0.0)}%
7967 \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<sub>T</sub><sub>E</sub>X, Pdf<sub>T</sub><sub>E</sub>X) or the `\shellescape` (X<sub>T</sub><sub>E</sub>X) 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.

```

7968 \ifx\pdfshellescape\undefined
7969 \ifx\shellescape\undefined
7970 \ifnum\markdownMode=0\relax
7971 \def\markdownExecuteShellEscape{1}%
7972 \else
7973 \def\markdownExecuteShellEscape{%
7974 \directlua{tex.sprint(status.shell_escape or "1")}}%
7975 \fi
7976 \else
7977 \let\markdownExecuteShellEscape\shellescape
7978 \fi
7979 \else
7980 \let\markdownExecuteShellEscape\pdfshellescape
7981 \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.

```

7982 \ifnum\markdownMode=0\relax
7983 \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
7984 \else
7985 \def\markdownExecuteDirect#1{%
7986 \directlua{os.execute("\luaescapestring{#1}")}}%
7987 \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.

```

7988 \def\markdownExecute#1{%
7989 \ifnum\markdownExecuteShellEscape=1\relax
7990 \markdownExecuteDirect{#1}%
7991 \else

```

```

7992 \markdownError{I can not access the shell}{Either run the TeX
7993 compiler with the --shell-escape or the --enable-write18 flag,
7994 or set shell_escape=t in the texmf.cnf file}%
7995 \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.

```

7996 \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.

```

7997 \catcode`\|=0%
7998 \catcode`\|=12%
7999 \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.

```

8000 |immediate|openout|markdownOutputFileStream=%
8001 |markdownOptionHelperScriptFileName
8002 |markdownInfo{Writing a helper Lua script to the file
8003 "|markdownOptionHelperScriptFileName"}%
8004 |immediate|write|markdownOutputFileStream{%
8005 local ran_ok, error = pcall(function()
8006 local ran_ok, kpse = pcall(require, "kpse")
8007 if ran_ok then kpse.set_program_name("luatex") end
8008 #1
8009 end)

```

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

```

8010 if not ran_ok then
8011 local file = io.open("%
8012 |markdownOptionOutputDir
8013 /|markdownOptionErrorTempFileName", "w")
8014 if file then
8015 file:write(error .. "\n")
8016 file:close()
8017 end
8018 print('\|markdownError{An error was encountered while executing
8019 Lua code}{For further clues, examine the file
8020 "|markdownOptionOutputDir
8021 /|markdownOptionErrorTempFileName}')
8022 end}%
8023 |immediate|closeout|markdownOutputFileStream

```

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

```
8024 |markdownInfo{Executing a helper Lua script from the file
8025 "|markdownOptionHelperScriptFileName" and storing the result in the
8026 file "|markdownOptionOutputTempFileName"}%
8027 |markdownExecute{texlua "|markdownOptionOutputDir
8028 /|markdownOptionHelperScriptFileName" > %
8029 "|markdownOptionOutputDir
8030 /|markdownOptionOutputTempFileName"}%
```

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

```
8031 |input|markdownOptionOutputTempFileName|relax}%
8032 |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.

```
8033 \fi
8034 \ifnum\markdownMode=2\relax
8035 \markdownWarning{Using mode 2: Direct Lua access
8036 (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,

```
8037 \beginingroup
```

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.

```
8038 \catcode`\|=0%
8039 \catcode`\|=12%
8040 |gdef|markdownLuaExecute#1{%
8041 |directlua{%
8042 local function print(input)
8043 local output = {}
8044 for line in input:gmatch("[^\r\n]+") do
8045 table.insert(output, line)
8046 end
8047 tex.print(output)
8048 end
8049 #1
```

```

8050 }%
8051 }%
8052 |endgroup
8053 \fi

```

### 3.2.7 Typesetting Markdown

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

```
8054 \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.

```

8055 \catcode`\|=0%
8056 \catcode`\|=12%
8057 \catcode`\&=6%
8058 |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.

```

8059 |begingroup
8060 |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.

```
8061 |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

8062 |markdownIfOption{frozenCache}{%
8063 |ifnum|markdownOptionFrozenCacheCounter=0|relax
8064 |markdownInfo{Reading frozen cache from
8065 "|markdownOptionFrozenCacheFileName"}%
8066 |input|markdownOptionFrozenCacheFileName|relax
8067 |fi
8068 |markdownInfo{Including markdown document number
8069 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
8070 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
8071 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8072 }-%
8073 |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  $\text{\LaTeX}$ Mk to track changes to the markdown document.

```

8074 |openin|markdownInputFileStream&1
8075 |closein|markdownInputFileStream
8076 |markdownPrepareLuaOptions
8077 |markdownLuaExecute{%
8078 |markdownPrepare
8079 local file = assert(io.open("&1", "r"),
8080 [[Could not open file "&1" for reading]])
8081 local input = assert(file:read("*a"))
8082 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.

```

8083 print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%

```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```

8084 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8085 }%
8086 |endgroup
8087 }%
8088 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\text{\TeX}$  to execute a  $\text{\TeX}$  document in the middle of a markdown document fragment.

```

8089 \gdef\markdownEscape#1{%
8090 \catcode`\%=14\relax
8091 \catcode`\#=6\relax
8092 \input #1\relax
8093 \catcode`\%=12\relax
8094 \catcode`\#=12\relax
8095 }%

```

### 3.3 $\text{\LaTeX}$ Implementation

The  $\text{\LaTeX}$  implementation makes use of the fact that, apart from some subtle differences,  $\text{\LaTeX}$  implements the majority of the plain  $\text{\TeX}$  format [12, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

8096 \def\markdownVersionSpace{ }%
8097 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
8098 \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.

```

8099 \ExplSyntaxOn
8100 \@@_latex_define_renderers:
8101 \@@_latex_define_renderer_prototypes:
8102 \ExplSyntaxOff

```

### 3.3.1 Logging Facilities

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

### 3.3.2 Typesetting Markdown

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

```

8103 \let\markdownInputPlainTeX\markdownInput
8104 \renewcommand\markdownInput[2] [] {%
8105 \begingroup
8106 \markdownSetup{#1}%
8107 \markdownInputPlainTeX{#2}%
8108 \endgroup}%

```

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

```

8109 \renewenvironment{markdown}{%
8110 \markdownReadAndConvert@markdown{}}{%
8111 \markdownEnd}%
8112 \renewenvironment{markdown*}[1]{%
8113 \markdownSetup{#1}%
8114 \markdownReadAndConvert@markdown*}{%
8115 \markdownEnd}%
8116 \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*.

```

8117 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
8118 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
8119 |gdef|markdownReadAndConvert@markdown#1<%
8120 |markdownReadAndConvert<\end{markdown#1}>%
8121 <|end<markdown#1>>>%
8122 |endgroup

```

**3.3.2.1 L<sup>A</sup>T<sub>E</sub>X Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
8123 \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.

```
8124 \newcommand\markdownLaTeXThemeName{}
8125 \seq_new:N \g_@@_latex_themes_seq
8126 \seq_gput_right:NV
8127 \g_@@_latex_themes_seq
8128 \markdownLaTeXThemeName
8129 \newcommand\markdownLaTeXThemeLoad[2]{
8130 \def\@tempa{%
8131 \def\markdownLaTeXThemeName{#2}
8132 \seq_gput_right:NV
8133 \g_@@_latex_themes_seq
8134 \markdownLaTeXThemeName
8135 \RequirePackage{#1}
8136 \seq_pop_right:NN
8137 \g_@@_latex_themes_seq
8138 \l_tmpa_tl
8139 \seq_get_right:NN
8140 \g_@@_latex_themes_seq
8141 \l_tmpa_tl
8142 \exp_args:NNV
8143 \def
8144 \markdownLaTeXThemeName
8145 \l_tmpa_tl}
8146 \ifmarkdownLaTeXLoaded
8147 \@tempa
8148 \else
8149 \exp_args:No
8150 \AtEndOfPackage
8151 { \@tempa }
8152 \fi}
8153 \ExplSyntaxOff
```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
8154 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
8155 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
8156 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
8157 \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:

```
8158 \renewcommand\markdownRendererInputFencedCode[2]{%
8159 \def\next##1 ##2\relax{%
8160 \ifthenelse{\equal{##1}{dot}}{%
8161 \markdownIfOption{frozenCache}{}{%
8162 \immediate\write18{%
8163 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
8164 then
8165 dot -Tpdf -o #1.pdf #1;
8166 cp #1 #1.pdf.source;
8167 fi}}%

```

We include the typeset image using the image token renderer:

```
8168 \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:

```
8169 }{%
8170 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
8171 }%
8172 }%
8173 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```
8174 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
8175 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```
8176 \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.

```
8177 \newcount\markdown@witiko@graphicx@http@counter
8178 \markdown@witiko@graphicx@http@counter=0
8179 \newcommand\markdown@witiko@graphicx@http@filename{%
8180 \markdownOptionCacheDir/witiko_graphicx_http%
8181 .\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.

```
8182 \newcommand\markdown@witiko@graphicx@http@download[2]{%
8183 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:

```
8184 \begingroup
8185 \catcode`\%=12
8186 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
8187 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
8188 \begingroup
8189 \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:

```
8190 \markdownIfOption{frozenCache}{}{^^A
8191 \immediate\write18{^^A
8192 mkdir -p "\markdownOptionCacheDir";
8193 if printf '%s' "#3" | grep -q -E '^https?:';
8194 then
```

The image will be downloaded to the pathname `\markdownOptionCacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
8195 OUTPUT_PREFIX="\markdownOptionCacheDir";
8196 OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
8197 OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
8198 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
8199 if ! [-e "$OUTPUT"];
8200 then
8201 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
8202 printf '%s' "$OUTPUT" > "\filename";
8203 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:

```
8204 else
8205 printf '%s' '#3' > "\filename";
8206 fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
8207 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
8208 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
8209 {#1}{#2}{\filename}{#4}^^A
8210 \endgroup
8211 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax^^A
8212 \endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
8213 \renewcommand\markdownRendererTildePrototype{~}%
```

### 3.3.3 Options

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

```
8214 \DeclareOption*{%
8215 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
8216 \ProcessOptions\relax
```

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

```
8217 \ExplSyntaxOn
8218 \keys_define:nn
8219 { markdown/latex-options }
8220 {
8221 renderers .code:n = {
8222 \keys_set:nn
8223 { markdown/latex-options/renderers }
8224 { #1 }
8225 },
8226 }
8227 \@_with_various_cases:nn
8228 { rendererPrototypes }
8229 {
8230 \keys_define:nn
8231 { markdown/latex-options }
8232 {
8233 #1 .code:n = {
8234 \keys_set:nn
8235 { markdown/latex-options/renderer-prototypes }
8236 { ##1 }
8237 },
8238 }
8239 }
```

The `code` key is used to immediately expand and execute code, which can be especially useful in  $\text{\LaTeX}$  setup snippets.

```
8240 \keys_define:nn
8241 { markdown/latex-options }
8242 {
8243 code .code:n = { #1 },
8244 }
```

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.

```

8245 \@@_with_various_cases:nn
8246 { jeekyllDataRenderers }
8247 {
8248 \keys_define:nn
8249 { markdown/latex-options }
8250 {
8251 #1 .code:n = {
8252 \tl_set:Nn
8253 \l_tmpa_tl
8254 { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the nput with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

8255 \tl_replace_all:NnV
8256 \l_tmpa_tl
8257 { / }
8258 \c_backslash_str
8259 \keys_set:nV
8260 { markdown/latex-options/jeekyll-data-renderers }
8261 \l_tmpa_tl
8262 },
8263 }
8264 }
8265 \keys_define:nn
8266 { markdown/latex-options/jeekyll-data-renderers }
8267 {
8268 unknown .code:n = {
8269 \tl_set_eq:NN
8270 \l_tmpa_tl
8271 \l_keys_key_str
8272 \tl_replace_all:NVn
8273 \l_tmpa_tl
8274 \c_backslash_str
8275 { / }
8276 \tl_put_right:Nn
8277 \l_tmpa_tl
8278 {
8279 .code:n = { #1 }
8280 }
8281 \keys_define:nV
8282 { markdown/jeekyllData }

```

```

8283 \l_tmpa_tl
8284 }
8285 }
8286 \cs_generate_variant:Nn
8287 \keys_define:nn
8288 { nV }
8289 \cs_generate_variant:Nn
8290 \tl_replace_all:Nnn
8291 { NVn }
8292 \cs_generate_variant:Nn
8293 \tl_replace_all:Nnn
8294 { NnV }
8295 \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.

```
8296 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

If the `tightLists` Lua option is disabled or the current document class is `beamer`, do not load the `paralist` package.

```

8297 \markdownIfOption{tightLists}{
8298 \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}%
8299 }{}

```

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.

```

8300 \ExplSyntaxOn
8301 \@ifpackageloaded{paralist}{
8302 \tl_new:N
8303 \l_@@_latex_fancy_list_item_label_number_style_tl
8304 \tl_new:N
8305 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8306 \cs_new:Nn
8307 \@@_latex_fancy_list_item_label_number:nn
8308 {
8309 \str_case:nn
8310 { #1 }
8311 {
8312 { Decimal } { #2 }
8313 { LowerRoman } { \int_to_roman:n { #2 } }
8314 { UpperRoman } { \int_to_Roman:n { #2 } }
8315 { LowerAlpha } { \int_to_alph:n { #2 } }
8316 { UpperAlpha } { \int_to_alph:n { #2 } }
8317 }

```

```

8318 }
8319 \cs_new:Nn
8320 \@@_latex_fancy_list_item_label_delimiter:n
8321 {
8322 \str_case:nn
8323 { #1 }
8324 {
8325 { Default } { . }
8326 { OneParen } {) }
8327 { Period } { . }
8328 }
8329 }
8330 \cs_new:Nn
8331 \@@_latex_fancy_list_item_label:nnn
8332 {
8333 \@@_latex_fancy_list_item_label_number:nn
8334 { #1 }
8335 { #3 }
8336 \@@_latex_fancy_list_item_label_delimiter:n
8337 { #2 }
8338 }
8339 \cs_new:Nn
8340 \@@_latex_paralist_style:nn
8341 {
8342 \str_case:nn
8343 { #1 }
8344 {
8345 { Decimal } { 1 }
8346 { LowerRoman } { i }
8347 { UpperRoman } { I }
8348 { LowerAlpha } { a }
8349 { UpperAlpha } { A }
8350 }
8351 \@@_latex_fancy_list_item_label_delimiter:n
8352 { #2 }
8353 }
8354 \markdownSetup{rendererPrototypes={
8355 ulBeginTight = {\begin{compactitem}},
8356 ulEndTight = {\end{compactitem}},
8357 fancyOlBegin = {
8358 \group_begin:
8359 \tl_set:Nn
8360 \l_@@_latex_fancy_list_item_label_number_style_tl
8361 { #1 }
8362 \tl_set:Nn
8363 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8364 { #2 }

```

```

8365 \tl_set:Nn
8366 \l_tmpa_tl
8367 { \begin{enumerate}[] }
8368 \tl_put_right:Nx
8369 \l_tmpa_tl
8370 { \@@_latex_paralist_style:nn { #1 } { #2 } }
8371 \tl_put_right:Nn
8372 \l_tmpa_tl
8373 {] }
8374 \l_tmpa_tl
8375 },
8376 fancyOlEnd = {
8377 \end{enumerate}
8378 \group_end:
8379 },
8380 olBeginTight = {\begin{compactenum}},
8381 olEndTight = {\end{compactenum}},
8382 fancyOlBeginTight = {
8383 \group_begin:
8384 \tl_set:Nn
8385 \l_@@_latex_fancy_list_item_label_number_style_tl
8386 { #1 }
8387 \tl_set:Nn
8388 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8389 { #2 }
8390 \tl_set:Nn
8391 \l_tmpa_tl
8392 { \begin{compactenum}[] }
8393 \tl_put_right:Nx
8394 \l_tmpa_tl
8395 { \@@_latex_paralist_style:nn { #1 } { #2 } }
8396 \tl_put_right:Nn
8397 \l_tmpa_tl
8398 {] }
8399 \l_tmpa_tl
8400 },
8401 fancyOlEndTight = {
8402 \end{compactenum}
8403 \group_end:
8404 },
8405 fancyOlItemWithNumber = {
8406 \item
8407 [
8408 \@@_latex_fancy_list_item_label:VVn
8409 \l_@@_latex_fancy_list_item_label_number_style_tl
8410 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8411 { #1 }

```

```

8412]
8413 },
8414 dlBeginTight = {\begin{compactdesc}},
8415 dlEndTight = {\end{compactdesc}}}}
8416 \cs_generate_variant:Nn
8417 \@@_latex_fancy_list_item_label:nnn
8418 { VVn }
8419 }{
8420 \markdownSetup{rendererPrototypes={
8421 ulBeginTight = {\markdownRendererUlBegin},
8422 ulEndTight = {\markdownRendererUlEnd},
8423 fancyOlBegin = {\markdownRendererOlBegin},
8424 fancyOlEnd = {\markdownRendererOlEnd},
8425 olBeginTight = {\markdownRendererOlBegin},
8426 olEndTight = {\markdownRendererOlEnd},
8427 fancyOlBeginTight = {\markdownRendererOlBegin},
8428 fancyOlEndTight = {\markdownRendererOlEnd},
8429 dlBeginTight = {\markdownRendererDlBegin},
8430 dlEndTight = {\markdownRendererDlEnd}}}
8431 }
8432 \ExplSyntaxOff
8433 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

8434 \ifpackageloaded{unicode-math}{
8435 \markdownSetup{rendererPrototypes={
8436 untickedBox = {\$ \mdlgwhtsquare$},
8437 }}
8438 }{
8439 \RequirePackage{amssymb}
8440 \markdownSetup{rendererPrototypes={
8441 untickedBox = {\$ \square$},
8442 }}
8443 }
8444 \RequirePackage{csvsimple}
8445 \RequirePackage{fancyvrb}
8446 \RequirePackage{graphicx}
8447 \markdownSetup{rendererPrototypes={
8448 lineBreak = {\},
8449 leftBrace = {\textbraceleft},
8450 rightBrace = {\textbraceright},
8451 dollarSign = {\textdollar},
8452 underscore = {\textunderscore},
8453 circumflex = {\textasciicircum},
8454 backslash = {\textbackslash},
8455 tilde = {\textasciitilde},

```

```
8456 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.

```
8457 codeSpan = {%
8458 \ifmmode
8459 \text{#1}%
8460 \else
8461 \texttt{#1}%
8462 \fi
8463 }}}}
8464 \ExplSyntaxOn
8465 \markdownSetup{
8466 rendererPrototypes = {
8467 contentBlock = {
8468 \str_case:nnF
8469 { #1 }
8470 {
8471 { csv }
8472 {
8473 \begin{table}
8474 \begin{center}
8475 \csvautotabular{#3}
8476 \end{center}
8477 \tl_if_empty:nF
8478 { #4 }
8479 { \caption{#4} }
8480 \end{table}
8481 }
8482 { tex } { \markdownEscape{#3} }
8483 }
8484 { \markdownInput{#3} }
8485 },
8486 },
8487 }
8488 \ExplSyntaxOff
8489 \markdownSetup{rendererPrototypes={
8490 image = {%
8491 \begin{figure}%
```

---

<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.



```

8492 \begin{center}%
8493 \includegraphics{#3}%
8494 \end{center}%
8495 \ifx\empty#4\empty\else
8496 \caption{#4}%
8497 \fi
8498 \end{figure}},
8499 ulBegin = {\begin{itemize}},
8500 ulEnd = {\end{itemize}},
8501 olBegin = {\begin{enumerate}},
8502 olItem = {\item{}},
8503 olItemWithNumber = {\item[#1.]},
8504 olEnd = {\end{enumerate}},
8505 dlBegin = {\begin{description}},
8506 dlItem = {\item[#1]},
8507 dlEnd = {\end{description}},
8508 emphasis = {\emph{#1}},
8509 tickedBox = {\\boxtimes},
8510 halfTickedBox = {\\boxdot},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

8511 headerAttributeContextBegin = {
8512 \markdownSetup{
8513 rendererPrototypes = {
8514 attributeIdentifier = {%
8515 \begingroup
8516 \def\next####1{%
8517 \def####1#####1{%
8518 \endgroup
8519 #####1{#####1}%
8520 \label{##1}%
8521 }%
8522 }%
8523 \next\markdownRendererHeadingOne
8524 \next\markdownRendererHeadingTwo
8525 \next\markdownRendererHeadingThree
8526 \next\markdownRendererHeadingFour
8527 \next\markdownRendererHeadingFive
8528 \next\markdownRendererHeadingSix
8529 },
8530 },
8531 }%
8532 },
8533 superscript = {#1},
8534 subscript = {\textsubscript{#1}},
8535 blockQuoteBegin = {\begin{quotation}},

```

```

8536 blockQuoteEnd = {\end{quotation}},
8537 inputVerbatim = {\VerbatimInput{#1}},
8538 inputFencedCode = {%
8539 \ifx\relax#2\relax
8540 \VerbatimInput{#1}%
8541 \else
8542 \@ifundefined{minted@code}{%
8543 \@ifundefined{lst@version}{%
8544 \markdownRendererInputFencedCode{#1}{}%

```

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

```

8545 }{%
8546 \lstinputlisting[language=#2]{#1}%
8547 }%

```

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

```

8548 }{%
8549 \catcode`\#=6\relax
8550 \inputminted{#2}{#1}%
8551 \catcode`\#=12\relax
8552 }%
8553 \fi},
8554 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
8555 note = {\footnote{#1}}}}

```

Support the nesting of strong emphasis.

```

8556 \ExplSyntaxOn
8557 \def\markdownLATEXStrongEmphasis#1{%
8558 \str_if_in:NnTF
8559 \f@series
8560 { b }
8561 { \textnormal{#1} }
8562 { \textbf{#1} }
8563 }
8564 \ExplSyntaxOff
8565 \markdownSetup{rendererPrototypes={strongEmphasis={%
8566 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

8567 \@ifundefined{chapter}{%
8568 \markdownSetup{rendererPrototypes = {
8569 headingOne = {\section{#1}},
8570 headingTwo = {\subsection{#1}},
8571 headingThree = {\subsubsection{#1}},
8572 headingFour = {\paragraph{#1}\leavevmode},
8573 headingFive = {\subparagraph{#1}\leavevmode}}}
8574 }{%
8575 \markdownSetup{rendererPrototypes = {

```

```

8576 headingOne = {\chapter{#1}},
8577 headingTwo = {\section{#1}},
8578 headingThree = {\subsection{#1}},
8579 headingFour = {\subsubsection{#1}},
8580 headingFive = {\paragraph{#1}\leavevmode},
8581 headingSix = {\subparagraph{#1}\leavevmode}}
8582 }%

```

**3.3.4.1 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

8583 \markdownSetup{
8584 rendererPrototypes = {
8585 ulItem = {%
8586 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
8587 },
8588 },
8589 }
8590 \def\markdownLaTeXUListItem{%
8591 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
8592 \item[\markdownLaTeXCheckbox]%
8593 \expandafter\@gobble
8594 \else
8595 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
8596 \item[\markdownLaTeXCheckbox]%
8597 \expandafter\expandafter\expandafter\@gobble
8598 \else
8599 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
8600 \item[\markdownLaTeXCheckbox]%
8601 \expandafter\expandafter\expandafter\expandafter
8602 \expandafter\expandafter\expandafter\@gobble
8603 \else
8604 \item{}%
8605 \fi
8606 \fi
8607 \fi
8608 }

```

**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.

```

8609 \@ifundefined{HCode}{}{
8610 \markdownSetup{
8611 rendererPrototypes = {
8612 inlineHtmlTag = {%
8613 \ifvmode

```

---

<sup>9</sup>See <https://tug.org/tex4ht/>.

```

8614 \IgnorePar
8615 \EndP
8616 \fi
8617 \HCode{#1}%
8618 },
8619 inputBlockHtmlElement = {%
8620 \ifvmode
8621 \IgnorePar
8622 \fi
8623 \EndP
8624 \special{t4ht*<#1}%
8625 \par
8626 \ShowPar
8627 },
8628 },
8629 }
8630 }

```

**3.3.4.3 Citations** Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

8631 \newcount\markdownLaTeXCitationsCounter
8632
8633 % Basic implementation
8634 \RequirePackage{gobble}
8635 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
8636 \advance\markdownLaTeXCitationsCounter by 1\relax
8637 \ifx\relax#4\relax
8638 \ifx\relax#5\relax
8639 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8640 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
8641 \expandafter\expandafter\expandafter
8642 \expandafter\expandafter\expandafter\expandafter
8643 \@gobblethree
8644 \fi
8645 \else% Before a postnote (#5), dump the accumulator
8646 \ifx\relax#1\relax\else
8647 \cite{#1}%
8648 \fi
8649 \cite[#5]{#6}%
8650 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8651 \else
8652 \expandafter\expandafter\expandafter
8653 \expandafter\expandafter\expandafter\expandafter
8654 \expandafter\expandafter\expandafter

```

```

8655 \expandafter\expandafter\expandafter\expandafter
8656 \markdownLaTeXBasicCitations
8657 \fi
8658 \expandafter\expandafter\expandafter
8659 \expandafter\expandafter\expandafter\expandafter{%
8660 \expandafter\expandafter\expandafter
8661 \expandafter\expandafter\expandafter\expandafter}%
8662 \expandafter\expandafter\expandafter
8663 \expandafter\expandafter\expandafter\expandafter{%
8664 \expandafter\expandafter\expandafter
8665 \expandafter\expandafter\expandafter\expandafter}%
8666 \expandafter\expandafter\expandafter
8667 \@gobblethree
8668 \fi
8669 \else% Before a prenote (#4), dump the accumulator
8670 \ifx\relax#1\relax\else
8671 \cite{#1}%
8672 \fi
8673 \ifnum\markdownLaTeXCitationsCounter>1\relax
8674 \space % Insert a space before the prenote in later citations
8675 \fi
8676 #4-\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
8677 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8678 \else
8679 \expandafter\expandafter\expandafter
8680 \expandafter\expandafter\expandafter\expandafter
8681 \markdownLaTeXBasicCitations
8682 \fi
8683 \expandafter\expandafter\expandafter{%
8684 \expandafter\expandafter\expandafter}%
8685 \expandafter\expandafter\expandafter{%
8686 \expandafter\expandafter\expandafter}%
8687 \expandafter
8688 \@gobblethree
8689 \fi\markdownLaTeXBasicCitations{#1#2#6},}
8690 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
8691
8692 % Natbib implementation
8693 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
8694 \advance\markdownLaTeXCitationsCounter by 1\relax
8695 \ifx\relax#3\relax
8696 \ifx\relax#4\relax
8697 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8698 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
8699 \expandafter\expandafter\expandafter
8700 \expandafter\expandafter\expandafter\expandafter
8701 \@gobbletwo

```

```

8702 \fi
8703 \else% Before a postnote (#4), dump the accumulator
8704 \ifx\relax#1\relax\else
8705 \citep{#1}%
8706 \fi
8707 \citep[] [#4]{#5}%
8708 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8709 \else
8710 \expandafter\expandafter\expandafter
8711 \expandafter\expandafter\expandafter\expandafter
8712 \expandafter\expandafter\expandafter
8713 \expandafter\expandafter\expandafter\expandafter
8714 \markdownLaTeXNatbibCitations
8715 \fi
8716 \expandafter\expandafter\expandafter
8717 \expandafter\expandafter\expandafter\expandafter{%
8718 \expandafter\expandafter\expandafter
8719 \expandafter\expandafter\expandafter\expandafter}%
8720 \expandafter\expandafter\expandafter
8721 \@gobbletwo
8722 \fi
8723 \else% Before a prenote (#3), dump the accumulator
8724 \ifx\relax#1\relax\relax\else
8725 \citep{#1}%
8726 \fi
8727 \citep[#3] [#4]{#5}%
8728 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8729 \else
8730 \expandafter\expandafter\expandafter
8731 \expandafter\expandafter\expandafter\expandafter
8732 \markdownLaTeXNatbibCitations
8733 \fi
8734 \expandafter\expandafter\expandafter{%
8735 \expandafter\expandafter\expandafter}%
8736 \expandafter
8737 \@gobbletwo
8738 \fi\markdownLaTeXNatbibCitations{#1,#5}}
8739 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
8740 \advance\markdownLaTeXCitationsCounter by 1\relax
8741 \ifx\relax#3\relax
8742 \ifx\relax#4\relax
8743 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8744 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
8745 \expandafter\expandafter\expandafter
8746 \expandafter\expandafter\expandafter\expandafter
8747 \@gobbletwo
8748 \fi

```

```

8749 \else% After a prenote or a postnote, dump the accumulator
8750 \ifx\relax#1\relax\else
8751 \citet{#1}%
8752 \fi
8753 , \citet[#3][#4]{#5}%
8754 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8755 ,
8756 \else
8757 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8758 ,
8759 \fi
8760 \fi
8761 \expandafter\expandafter\expandafter
8762 \expandafter\expandafter\expandafter\expandafter
8763 \markdownLaTeXNatbibTextCitations
8764 \expandafter\expandafter\expandafter
8765 \expandafter\expandafter\expandafter\expandafter{%
8766 \expandafter\expandafter\expandafter
8767 \expandafter\expandafter\expandafter\expandafter}%
8768 \expandafter\expandafter\expandafter
8769 \@gobbletwo
8770 \fi
8771 \else% After a prenote or a postnote, dump the accumulator
8772 \ifx\relax#1\relax\relax\else
8773 \citet{#1}%
8774 \fi
8775 , \citet[#3][#4]{#5}%
8776 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8777 ,
8778 \else
8779 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8780 ,
8781 \fi
8782 \fi
8783 \expandafter\expandafter\expandafter
8784 \markdownLaTeXNatbibTextCitations
8785 \expandafter\expandafter\expandafter{%
8786 \expandafter\expandafter\expandafter}%
8787 \expandafter
8788 \@gobbletwo
8789 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
8790
8791 % BibLaTeX implementation
8792 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
8793 \advance\markdownLaTeXCitationsCounter by 1\relax
8794 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8795 \autocites#1[#3][#4]{#5}%

```

```

8796 \expandafter\@gobbletwo
8797 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
8798 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
8799 \advance\markdownLaTeXCitationsCounter by 1\relax
8800 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8801 \textcites#1[#3][#4]{#5}%
8802 \expandafter\@gobbletwo
8803 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
8804
8805 \markdownSetup{rendererPrototypes = {
8806 cite = {%
8807 \markdownLaTeXCitationsCounter=1%
8808 \def\markdownLaTeXCitationsTotal{#1}%
8809 \@ifundefined{autocites}{%
8810 \@ifundefined{citep}{%
8811 \expandafter\expandafter\expandafter
8812 \markdownLaTeXBasicCitations
8813 \expandafter\expandafter\expandafter{%
8814 \expandafter\expandafter\expandafter}%
8815 \expandafter\expandafter\expandafter{%
8816 \expandafter\expandafter\expandafter}%
8817 }{%
8818 \expandafter\expandafter\expandafter
8819 \markdownLaTeXNatbibCitations
8820 \expandafter\expandafter\expandafter{%
8821 \expandafter\expandafter\expandafter}%
8822 }%
8823 }{%
8824 \expandafter\expandafter\expandafter
8825 \markdownLaTeXBibLaTeXCitations
8826 \expandafter{\expandafter}%
8827 }},
8828 textCite = {%
8829 \markdownLaTeXCitationsCounter=1%
8830 \def\markdownLaTeXCitationsTotal{#1}%
8831 \@ifundefined{autocites}{%
8832 \@ifundefined{citep}{%
8833 \expandafter\expandafter\expandafter
8834 \markdownLaTeXBasicTextCitations
8835 \expandafter\expandafter\expandafter{%
8836 \expandafter\expandafter\expandafter}%
8837 \expandafter\expandafter\expandafter{%
8838 \expandafter\expandafter\expandafter}%
8839 }{%
8840 \expandafter\expandafter\expandafter
8841 \markdownLaTeXNatbibTextCitations
8842 \expandafter\expandafter\expandafter{%

```



```

8843 \expandafter\expandafter\expandafter}%
8844 }%
8845 }{
8846 \expandafter\expandafter\expandafter
8847 \markdownLaTeXBibLaTeXTextCitations
8848 \expandafter{\expandafter}%
8849 }}}

```

**3.3.4.4 Links** Before consuming the parameters for the hyperlink renderer, we change the category of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

8850 \RequirePackage{url}
8851 \RequirePackage{expl3}
8852 \ExplSyntaxOn
8853 \def\markdownRendererLinkPrototype#1#2#3#4{
8854 \tl_set:Nn \l_tmpa_tl { #1 }
8855 \tl_set:Nn \l_tmpb_tl { #2 }
8856 \bool_set:Nn
8857 \l_tmpa_bool
8858 {
8859 \tl_if_eq_p:NN
8860 \l_tmpa_tl
8861 \l_tmpb_tl
8862 }
8863 \tl_set:Nn \l_tmpa_tl { #4 }
8864 \bool_set:Nn
8865 \l_tmpb_bool
8866 {
8867 \tl_if_empty_p:N
8868 \l_tmpa_tl
8869 }

```

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.

```

8870 \bool_if:nTF
8871 {
8872 \l_tmpa_bool && \l_tmpb_bool
8873 }
8874 {
8875 \markdownLaTeXRendererAutolink { #2 } { #3 }
8876 }{
8877 \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
8878 }
8879 }
8880 \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.

```

8881 \tl_set:Nn
8882 \l_tmpa_tl
8883 { #2 }
8884 \tl_trim_spaces:N
8885 \l_tmpa_tl
8886 \tl_set:Nx
8887 \l_tmpb_tl
8888 {
8889 \tl_range:Nnn
8890 \l_tmpa_tl
8891 { 1 }
8892 { 1 }
8893 }
8894 \str_if_eq:NNTF
8895 \l_tmpb_tl
8896 \c_hash_str
8897 {
8898 \tl_set:Nx
8899 \l_tmpb_tl
8900 {
8901 \tl_range:Nnn
8902 \l_tmpa_tl
8903 { 2 }
8904 { -1 }
8905 }
8906 \exp_args:NV
8907 \ref
8908 \l_tmpb_tl
8909 }{
8910 \url { #2 }
8911 }
8912 }
8913 \ExplSyntaxOff
8914 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
8915 #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.

```

8916 \newcount\markdownLaTeXRowCount
8917 \newcount\markdownLaTeXRowTotal
8918 \newcount\markdownLaTeXColumnCounter
8919 \newcount\markdownLaTeXColumnTotal
8920 \newtoks\markdownLaTeXTable
8921 \newtoks\markdownLaTeXTableAlignment
```

```

8922 \newtoks\markdownLaTeXTableEnd
8923 \AtBeginDocument{%
8924 \@ifpackageloaded{booktabs}{%
8925 \def\markdownLaTeXTopRule{\toprule}%
8926 \def\markdownLaTeXMidRule{\midrule}%
8927 \def\markdownLaTeXBottomRule{\bottomrule}%
8928 }{%
8929 \def\markdownLaTeXTopRule{\hline}%
8930 \def\markdownLaTeXMidRule{\hline}%
8931 \def\markdownLaTeXBottomRule{\hline}%
8932 }%
8933 }
8934 \markdownSetup{rendererPrototypes={
8935 table = {%
8936 \markdownLaTeXTable={}%
8937 \markdownLaTeXTableAlignment={}%
8938 \markdownLaTeXTableEnd={%
8939 \markdownLaTeXBottomRule
8940 \end{tabular}}}%
8941 \ifx\empty#1\empty\else
8942 \addto@hook\markdownLaTeXTable{%
8943 \begin{table}
8944 \centering}%
8945 \addto@hook\markdownLaTeXTableEnd{%
8946 \caption{#1}
8947 \end{table}}}%
8948 \fi
8949 \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
8950 \markdownLaTeXRowCounter=0%
8951 \markdownLaTeXRowTotal=#2%
8952 \markdownLaTeXColumnTotal=#3%
8953 \markdownLaTeXRenderTableRow
8954 }
8955 }}
8956 \def\markdownLaTeXRenderTableRow#1{%
8957 \markdownLaTeXColumnCounter=0%
8958 \ifnum\markdownLaTeXRowCounter=0\relax
8959 \markdownLaTeXReadAlignments#1%
8960 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
8961 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
8962 \the\markdownLaTeXTableAlignment}}}%
8963 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
8964 \else
8965 \markdownLaTeXRenderTableCell#1%
8966 \fi
8967 \ifnum\markdownLaTeXRowCounter=1\relax
8968 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule

```

```

8969 \fi
8970 \advance\markdownLaTeXRowCount by 1\relax
8971 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
8972 \the\markdownLaTeXTable
8973 \the\markdownLaTeXTableEnd
8974 \expandafter\@gobble
8975 \fi\markdownLaTeXRenderTableRow}
8976 \def\markdownLaTeXReadAlignments#1{%
8977 \advance\markdownLaTeXColumnCounter by 1\relax
8978 \if#1d%
8979 \addto@hook\markdownLaTeXTableAlignment{1}%
8980 \else
8981 \addto@hook\markdownLaTeXTableAlignment{#1}%
8982 \fi
8983 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
8984 \expandafter\@gobble
8985 \fi\markdownLaTeXReadAlignments}
8986 \def\markdownLaTeXRenderTableCell#1{%
8987 \advance\markdownLaTeXColumnCounter by 1\relax
8988 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
8989 \addto@hook\markdownLaTeXTable{#1&}%
8990 \else
8991 \addto@hook\markdownLaTeXTable{#1\\}%
8992 \expandafter\@gobble
8993 \fi\markdownLaTeXRenderTableCell}

```

**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.

```

8994 \ExplSyntaxOn
8995 \keys_define:nn
8996 { markdown/jekyllData }
8997 {
8998 author .code:n = { \author{#1} },
8999 date .code:n = { \date{#1} },
9000 title .code:n = { \title{#1} },
9001 }

```

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.

```

9002 % TODO: Remove the command definition in TeX Live 2021.
9003 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
9004 \markdownSetup{
9005 rendererPrototypes = {

```

```

9006 jekyllDataEnd = {
9007 % TODO: Remove the else branch in TeX Live 2021.
9008 \IfFormatAtLeastTF
9009 { 2020-10-01 }
9010 { \AddToHook{begindocument/end}{\maketitle} }
9011 {
9012 \ifx\@onlypreamble\@notprerr
9013 % We are in the document
9014 \maketitle
9015 \else
9016 % We are in the preamble
9017 \RequirePackage{etoolbox}
9018 \AfterEndPreamble{\maketitle}
9019 \fi
9020 }
9021 },
9022 },
9023 }
9024 \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.

```

9025 \markdownIfOption{strikeThrough}{%
9026 \RequirePackage{soulutf8}%
9027 \markdownSetup{
9028 rendererPrototypes = {
9029 strikeThrough = {%
9030 \st{#1}%
9031 },
9032 }
9033 }
9034 }{}

```

**3.3.4.8 Raw Attribute Renderer Prototypes** In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex` or `latex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

9035 \ExplSyntaxOn
9036 \cs_gset:Npn
9037 \markdownRendererInputRawInlinePrototype#1#2
9038 {
9039 \str_case:nn
9040 { #2 }
9041 {
9042 { tex } { \markdownEscape{#1} }

```

```

9043 { latex } { \markdownEscape{#1} }
9044 { md } { \markdownInput{#1} }
9045 }
9046 }
9047 \cs_gset_eq:NN
9048 \markdownRendererInputRawBlockPrototype
9049 \markdownRendererInputRawInlinePrototype
9050 \ExplSyntaxOff
9051 \fi % Closes ` \markdownIfOption{Plain}{\iffalse}{iftrue}`

```

### 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.

```

9052 \newcommand\markdownMakeOther{%
9053 \count0=128\relax
9054 \loop
9055 \catcode\count0=11\relax
9056 \advance\count0 by 1\relax
9057 \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

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

```

9058 \def\markdownMakeOther{%
9059 \count0=128\relax
9060 \loop
9061 \catcode\count0=11\relax
9062 \advance\count0 by 1\relax
9063 \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 ConTeXt.

```

9064 \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
9065 \long\def\inputmarkdown{%
9066 \dosingleempty
9067 \doinputmarkdown}%
9068 \long\def\doinputmarkdown[#1]#2{%
9069 \begingroup
9070 \iffirstargument
9071 \setupmarkdown{#1}%
9072 \fi
9073 \markdownInput{#2}%
9074 \endgroup}%
```

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

In Knuth's T<sub>E</sub>X, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, 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)T<sub>E</sub>X, but ConT<sub>E</sub>Xt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT<sub>E</sub>Xt MkIV and therefore to insert hard line breaks into markdown text.

```
9075 \ifx\startluacode\undefined % MkII
9076 \begingroup
9077 \catcode`\|=0%
9078 \catcode`\|=12%
9079 |gdef|startmarkdown{%
9080 |markdownReadAndConvert{\stopmarkdown}%
9081 |stopmarkdown}}%
9082 |gdef|stopmarkdown{%
9083 |markdownEnd}%
9084 |endgroup
9085 \else % MkIV
9086 \startluacode
9087 document.markdown_buffering = false
9088 local function preserve_trailing_spaces(line)
9089 if document.markdown_buffering then
9090 line = line:gsub("[\t][\t]$", "\t\t")
9091 end
9092 return line
9093 end
9094 resolvers.installinputlinehandler(preserve_trailing_spaces)
9095 \stoptluacode
9096 \begingroup
9097 \catcode`\|=0%
9098 \catcode`\|=12%
```

```

9099 |gdef|startmarkdown{%
9100 |ctxlua{document.markdown_buffering = true}%
9101 |markdownReadAndConvert{\stopmarkdown}%
9102 |stopmarkdown}}%
9103 |gdef|stopmarkdown{%
9104 |ctxlua{document.markdown_buffering = false}%
9105 |markdownEnd}%
9106 |endgroup
9107 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

9108 \def\markdownRendererLineBreakPrototype{\blank}%
9109 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
9110 \def\markdownRendererRightBracePrototype{\textbraceright}%
9111 \def\markdownRendererDollarSignPrototype{\textdollar}%
9112 \def\markdownRendererPercentSignPrototype{\percent}%
9113 \def\markdownRendererUnderscorePrototype{\textunderscore}%
9114 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
9115 \def\markdownRendererBackslashPrototype{\textbackslash}%
9116 \def\markdownRendererTildePrototype{\textasciitilde}%
9117 \def\markdownRendererPipePrototype{\char`|}%
9118 \def\markdownRendererLinkPrototype#1#2#3#4{%
9119 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
9120 \fi\texttt<\hyphenatedurl{#3}>}}%
9121 \usemodule[database]
9122 \defineseparatedlist
9123 [MarkdownConTeXtCSV]
9124 [separator={,},
9125 before=\bTABLE,after=\eTABLE,
9126 first=\bTR,last=\eTR,
9127 left=\bTD,right=\eTD]
9128 \def\markdownConTeXtCSV{csv}
9129 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
9130 \def\markdownConTeXtCSV@arg{#1}%
9131 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
9132 \placetable[] [tab:#1]{#4}{%
9133 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
9134 \else
9135 \markdownInput{#3}%
9136 \fi}%
9137 \def\markdownRendererImagePrototype#1#2#3#4{%
9138 \placefigure[] []{#4}{\externalfigure[#3]}}%
9139 \def\markdownRendererULBeginPrototype{\startitemize}%
9140 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
9141 \def\markdownRendererULItemPrototype{\item}%

```



```

9142 \def\markdownRendererU1EndPrototype{\stopitemize}%
9143 \def\markdownRendererU1EndTightPrototype{\stopitemize}%
9144 \def\markdownRendererO1BeginPrototype{\startitemize[n]}%
9145 \def\markdownRendererO1BeginTightPrototype{\startitemize[packed,n]}%
9146 \def\markdownRendererO1ItemPrototype{\item}%
9147 \def\markdownRendererO1ItemWithNumberPrototype#1{\sym{#1.}}%
9148 \def\markdownRendererO1EndPrototype{\stopitemize}%
9149 \def\markdownRendererO1EndTightPrototype{\stopitemize}%
9150 \definedescription
9151 [MarkdownConTeXtDlItemPrototype]
9152 [location=hanging,
9153 margin=standard,
9154 headstyle=bold]%
9155 \definestartstop
9156 [MarkdownConTeXtDlPrototype]
9157 [before=\blank,
9158 after=\blank]%
9159 \definestartstop
9160 [MarkdownConTeXtDlTightPrototype]
9161 [before=\blank\startpacked,
9162 after=\stoppacked\blank]%
9163 \def\markdownRendererDlBeginPrototype{%
9164 \startMarkdownConTeXtDlPrototype}%
9165 \def\markdownRendererDlBeginTightPrototype{%
9166 \startMarkdownConTeXtDlTightPrototype}%
9167 \def\markdownRendererDlItemPrototype#1{%
9168 \startMarkdownConTeXtDlItemPrototype{#1}}%
9169 \def\markdownRendererDlItemEndPrototype{%
9170 \stopMarkdownConTeXtDlItemPrototype}%
9171 \def\markdownRendererDlEndPrototype{%
9172 \stopMarkdownConTeXtDlPrototype}%
9173 \def\markdownRendererDlEndTightPrototype{%
9174 \stopMarkdownConTeXtDlTightPrototype}%
9175 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
9176 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
9177 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
9178 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
9179 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
9180 \def\markdownRendererInputFencedCodePrototype#1#2{%
9181 \ifx\relax#2\relax
9182 \typefile{#1}%
9183 \else

```

The code fence infostring is used as a name from the ConTeXt `\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

```

```

9184 \typefile[#2] []{#1}%
9185 \fi}%
9186 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
9187 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
9188 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
9189 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
9190 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
9191 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
9192 \def\markdownRendererThematicBreakPrototype{%
9193 \blackrule[height=1pt, width=\hsize]}%
9194 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
9195 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
9196 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
9197 \def\markdownRendererUntickedBoxPrototype{${\square}$}
9198 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
9199 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
9200 \def\markdownRendererSubscriptPrototype#1{\low{#1}}

```

#### 3.4.2.1 Tables

There is a basic implementation of tables.

```

9201 \newcount\markdownConTeXtRowCounter
9202 \newcount\markdownConTeXtRowTotal
9203 \newcount\markdownConTeXtColumnCounter
9204 \newcount\markdownConTeXtColumnTotal
9205 \newtoks\markdownConTeXtTable
9206 \newtoks\markdownConTeXtTableFloat
9207 \def\markdownRendererTablePrototype#1#2#3{%
9208 \markdownConTeXtTable={}%
9209 \ifx\empty#1\empty
9210 \markdownConTeXtTableFloat={%
9211 \the\markdownConTeXtTable}%
9212 \else
9213 \markdownConTeXtTableFloat={%
9214 \placetable{#1}{\the\markdownConTeXtTable}}%

```

```

9215 \fi
9216 \begingroup
9217 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9218 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9219 \setupTABLE[r][1][topframe=on, bottomframe=on]
9220 \setupTABLE[r][#1][bottomframe=on]
9221 \markdownConTeXtRowCounter=0%
9222 \markdownConTeXtRowTotal=#2%
9223 \markdownConTeXtColumnTotal=#3%
9224 \markdownConTeXtRenderTableRow}
9225 \def\markdownConTeXtRenderTableRow#1{%
9226 \markdownConTeXtColumnCounter=0%
9227 \ifnum\markdownConTeXtRowCounter=0\relax
9228 \markdownConTeXtReadAlignments#1%
9229 \markdownConTeXtTable={\bTABLE}%
9230 \else
9231 \markdownConTeXtTable=\expandafter{%
9232 \the\markdownConTeXtTable\bTR}%
9233 \markdownConTeXtRenderTableCell#1%
9234 \markdownConTeXtTable=\expandafter{%
9235 \the\markdownConTeXtTable\eTR}%
9236 \fi
9237 \advance\markdownConTeXtRowCounter by 1\relax
9238 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
9239 \markdownConTeXtTable=\expandafter{%
9240 \the\markdownConTeXtTable\eTABLE}%
9241 \the\markdownConTeXtTableFloat
9242 \endgroup
9243 \expandafter\gobbleoneargument
9244 \fi\markdownConTeXtRenderTableRow}
9245 \def\markdownConTeXtReadAlignments#1{%
9246 \advance\markdownConTeXtColumnCounter by 1\relax
9247 \if#1d%
9248 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9249 \fi\if#1l%
9250 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9251 \fi\if#1c%
9252 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
9253 \fi\if#1r%
9254 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
9255 \fi
9256 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9257 \expandafter\gobbleoneargument
9258 \fi\markdownConTeXtReadAlignments}
9259 \def\markdownConTeXtRenderTableCell#1{%
9260 \advance\markdownConTeXtColumnCounter by 1\relax
9261 \markdownConTeXtTable=\expandafter{%

```

```

9262 \the\markdownConTeXtTable\bTD#1\eTD}%
9263 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9264 \expandafter\gobbleoneargument
9265 \fi\markdownConTeXtRenderTableCell}

```

**3.4.2.2 Raw Attribute Renderer Prototypes** In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex` or `context`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

9266 \ExplSyntaxOn
9267 \cs_gset:Npn
9268 \markdownRendererInputRawInlinePrototype#1#2
9269 {
9270 \str_case:nn
9271 { #2 }
9272 {
9273 { tex } { \markdownEscape{#1} }
9274 { context } { \markdownEscape{#1} }
9275 { md } { \markdownInput{#1} }
9276 }
9277 }
9278 \cs_gset_eq:NN
9279 \markdownRendererInputRawBlockPrototype
9280 \markdownRendererInputRawInlinePrototype
9281 \ExplSyntaxOff
9282 \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] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] 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.
- [7] 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).
- [8] 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).
- [9] 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).
- [10] 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).
- [11] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] 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).
- [13] 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.
- [14] 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>	268
<code>\autocites</code>	260
<code>blankBeforeBlockquote</code>	16
<code>blankBeforeCodeFence</code>	17
<code>blankBeforeHeading</code>	17
<code>breakableBlockquotes</code>	17
<code>cacheDir</code>	15, 21, 44, 45, 196
<code>citationNbsps</code>	18
<code>citations</code>	18, 76, 77
<code>\cite</code>	260

<code>\citep</code>	260
<code>\citet</code>	260
<code>codeSpans</code>	19
<code>compactdesc</code>	4
<code>compactenum</code>	4
<code>compactitem</code>	4
<code>contentBlocks</code>	15, 19
<code>contentBlocksLanguageMap</code>	15
<code>convert</code>	236
<code>\csvautotabular</code>	4
<code>\date</code>	268
<code>debugExtensions</code>	8, 15, 20, 192
<code>debugExtensionsFileName</code>	15, 20
<code>defaultOptions</code>	9, 40, 223
<code>\definetyping</code>	273
<code>definitionLists</code>	20, 64
<code>\directlua</code>	4, 45, 243
<code>eagerCache</code>	21, 21
<code>\enableregime</code>	270
<code>\endmarkdown</code>	87
<code>entities.char_entity</code>	155
<code>entities.dec_entity</code>	155
<code>entities.hex_entity</code>	155
<code>escape_citation</code>	197, 198
<code>escaped_citation_chars</code>	197, 197
<code>expandtabs</code>	178
<code>expectJekyllData</code>	23
<code>extensions</code>	21, 106, 197
<code>extensions.citations</code>	197
<code>extensions.content_blocks</code>	201
<code>extensions.definition_lists</code>	204
<code>extensions.fancy_lists</code>	219
<code>extensions.fenced_code</code>	206
<code>extensions.header_attributes</code>	208
<code>extensions.jekyll_data</code>	210
<code>extensions.notes</code>	207
<code>extensions.pipe_table</code>	213
<code>extensions.raw_attribute</code>	216
<code>extensions.strike_through</code>	217
<code>extensions.subscripts</code>	219

<code>extensions.superscripts</code>	218
<code>fancyLists</code>	24, 59–64
<code>fencedCode</code>	24, 31, 68, 82, 247
<code>\filecontents</code>	238
<code>finalizeCache</code>	16, 21, 25, 26, 44, 196
<code>frozenCacheCounter</code>	26, 196, 244, 245
<code>frozenCacheFileName</code>	16, 25, 196
<code>\g_luabridge_error_output_filename_str</code>	46
<code>\g_luabridge_helper_script_filename_str</code>	45
<code>hardLineBreaks</code>	27
<code>hashEnumerators</code>	27
<code>headerAttributes</code>	27, 32, 79, 80
<code>html</code>	28, 77, 78, 259
<code>hybrid</code>	28, 31, 36, 37, 46, 52, 94, 158, 179, 244
<code>\includegraphics</code>	4
<code>inlineNotes</code>	29
<code>\input</code>	42, 103, 239, 243
<code>\inputmarkdown</code>	103, 103, 104, 271
<code>isdir</code>	3
<code>iterlines</code>	178
<code>jekyllData</code>	3, 23, 29, 69–72
<code>\jobname</code>	45, 46
<code>\label</code>	257
<code>languages_json</code>	201, 201
<code>\maketitle</code>	268
<code>\markdown</code>	87
<code>markdown</code>	87, 87, 246
<code>markdown*</code>	87, 87, 88, 246
<code>\markdown_jekyll_data_concatenate_address:NN</code>	232
<code>\markdown_jekyll_data_pop:</code>	233
<code>\markdown_jekyll_data_push:nN</code>	233
<code>\markdown_jekyll_data_push_address_segment:n</code>	231
<code>\markdown_jekyll_data_set_keyval:Nn</code>	233
<code>\markdown_jekyll_data_set_keyvals:nn</code>	233
<code>\markdown_jekyll_data_update_address_tls:</code>	232
<code>\markdownBegin</code>	42, 42, 43, 85, 87, 103
<code>\markdownEnd</code>	42, 42, 43, 85, 87, 103

<code>\markdownError</code>	84, 84
<code>\markdownEscape</code>	42, 43, 245
<code>\markdownExecute</code>	241
<code>\markdownExecuteDirect</code>	241, 241
<code>\markdownExecuteShellEscape</code>	241, 241
<code>\markdownIfOption</code>	237
<code>\markdownIfSnippetExists</code>	89
<code>\markdownInfo</code>	84
<code>\markdownInput</code>	42, 43, 87, 88, 103, 244, 246
<code>\markdownInputFileStream</code>	237
<code>\markdownInputPlainTeX</code>	246
<code>\markdownLuaExecute</code>	240, 242, 243, 244
<code>\markdownLuaOptions</code>	234, 236
<code>\markdownLuaRegisterIBCallback</code>	86
<code>\markdownLuaUnregisterIBCallback</code>	86
<code>\markdownMakeOther</code>	84, 270
<code>\markdownMode</code>	4, 45, 46, 85, 85, 240, 243
<code>\markdownOptionCacheDir</code>	3, 99, 236, 249
<code>\markdownOptionErrorTempFileName</code>	46, 46, 242
<code>\markdownOptionFinalizeCache</code>	44, 44, 99
<code>\markdownOptionFrozenCache</code>	16, 25, 44, 45, 93, 94, 99, 248, 249
<code>\markdownOptionFrozenCacheFileName</code>	44
<code>\markdownOptionHelperScriptFileName</code>	45, 45–47, 242, 243
<code>\markdownOptionHybrid</code>	46, 99
<code>\markdownOptionInputTempFileName</code>	45, 238, 239
<code>\markdownOptionOutputDir</code>	46
<code>\markdownOptionOutputTempFileName</code>	45, 46, 243
<code>\markdownOptionSmartEllipses</code>	99
<code>\markdownOptionStripPercentSigns</code>	48, 238
<code>\markdownOutputFileStream</code>	237
<code>\markdownPrepare</code>	236
<code>\markdownPrepareLuaOptions</code>	234
<code>\markdownReadAndConvert</code>	85, 238, 246, 271
<code>\markdownReadAndConvertProcessLine</code>	239, 240
<code>\markdownReadAndConvertStripPercentSigns</code>	238
<code>\markdownReadAndConvertTab</code>	237
<code>\markdownRendererAttributeClassName</code>	79
<code>\markdownRendererAttributeIdentifier</code>	79
<code>\markdownRendererAttributeKeyValue</code>	79
<code>\markdownRendererBlockHtmlCommentBegin</code>	78
<code>\markdownRendererBlockHtmlCommentEnd</code>	78
<code>\markdownRendererBlockQuoteBegin</code>	67



<code>\markdownRendererBlockQuoteEnd</code>	68
<code>\markdownRendererCite</code>	76, 77
<code>\markdownRendererCodeSpan</code>	55
<code>\markdownRendererCodeSpanPrototype</code>	102
<code>\markdownRendererContentBlock</code>	56, 56
<code>\markdownRendererContentBlockCode</code>	56
<code>\markdownRendererContentBlockOnlineImage</code>	56
<code>\markdownRendererDlBegin</code>	64
<code>\markdownRendererDlBeginTight</code>	64
<code>\markdownRendererDlDefinitionBegin</code>	65
<code>\markdownRendererDlDefinitionEnd</code>	66
<code>\markdownRendererDlEnd</code>	66
<code>\markdownRendererDlEndTight</code>	66
<code>\markdownRendererDlItem</code>	65
<code>\markdownRendererDlItemEnd</code>	65
<code>\markdownRendererDocumentBegin</code>	50
<code>\markdownRendererDocumentEnd</code>	50
<code>\markdownRendererEllipsis</code>	33, 51
<code>\markdownRendererEmphasis</code>	67, 101
<code>\markdownRendererFancyOlBegin</code>	60, 60
<code>\markdownRendererFancyOlBeginTight</code>	60
<code>\markdownRendererFancyOlEnd</code>	63
<code>\markdownRendererFancyOlEndTight</code>	64
<code>\markdownRendererFancyOlItem</code>	62
<code>\markdownRendererFancyOlItemEnd</code>	62
<code>\markdownRendererFancyOlItemWithNumber</code>	62
<code>\markdownRendererFootnote</code>	75, 83
<code>\markdownRendererFootnotePrototype</code>	75, 83
<code>\markdownRendererHalfTickedBox</code>	49
<code>\markdownRendererHeaderAttributeContextBegin</code>	80
<code>\markdownRendererHeaderAttributeContextEnd</code>	80
<code>\markdownRendererHeadingFive</code>	73
<code>\markdownRendererHeadingFour</code>	73
<code>\markdownRendererHeadingOne</code>	72
<code>\markdownRendererHeadingSix</code>	74
<code>\markdownRendererHeadingThree</code>	73
<code>\markdownRendererHeadingTwo</code>	72
<code>\markdownRendererHorizontalRule</code>	74, 83
<code>\markdownRendererHorizontalRulePrototype</code>	74, 83
<code>\markdownRendererImage</code>	55
<code>\markdownRendererImagePrototype</code>	102
<code>\markdownRendererInlineHtmlComment</code>	77

<code>\markdownRendererInlineHtmlTag</code>	78
<code>\markdownRendererInputBlockHtmlElement</code>	78
<code>\markdownRendererInputFencedCode</code>	68
<code>\markdownRendererInputRawBlock</code>	82
<code>\markdownRendererInputRawInline</code>	82
<code>\markdownRendererInputVerbatim</code>	68
<code>\markdownRendererInterblockSeparator</code>	51
<code>\markdownRendererJekyllDataBegin</code>	69
<code>\markdownRendererJekyllDataBoolean</code>	71
<code>\markdownRendererJekyllDataEmpty</code>	72
<code>\markdownRendererJekyllDataEnd</code>	69
<code>\markdownRendererJekyllDataMappingBegin</code>	69
<code>\markdownRendererJekyllDataMappingEnd</code>	70
<code>\markdownRendererJekyllDataNumber</code>	71
<code>\markdownRendererJekyllDataSequenceBegin</code>	70
<code>\markdownRendererJekyllDataSequenceEnd</code>	70
<code>\markdownRendererJekyllDataString</code>	71
<code>\markdownRendererLineBreak</code>	51
<code>\markdownRendererLink</code>	55, 101
<code>\markdownRendererNbsp</code>	52
<code>\markdownRendererNote</code>	75
<code>\markdownRendererOlBegin</code>	59
<code>\markdownRendererOlBeginTight</code>	59
<code>\markdownRendererOlEnd</code>	63
<code>\markdownRendererOlEndTight</code>	63
<code>\markdownRendererOlItem</code>	33, 61
<code>\markdownRendererOlItemEnd</code>	61
<code>\markdownRendererOlItemWithNumber</code>	33, 61
<code>\markdownRendererStrikeThrough</code>	81
<code>\markdownRendererStrongEmphasis</code>	67
<code>\markdownRendererSubscript</code>	81
<code>\markdownRendererSuperscript</code>	81
<code>\markdownRendererTable</code>	77
<code>\markdownRendererTextCite</code>	77
<code>\markdownRendererThematicBreak</code>	74
<code>\markdownRendererTickedBox</code>	49
<code>\markdownRendererUlBegin</code>	57
<code>\markdownRendererUlBeginTight</code>	57
<code>\markdownRendererUlEnd</code>	58
<code>\markdownRendererUlEndTight</code>	59
<code>\markdownRendererUlItem</code>	58
<code>\markdownRendererUlItemEnd</code>	58

<code>\markdownRendererUntickedBox</code>	<i>49</i>
<code>\markdownSetup</code>	<i>88, 88, 245, 250</i>
<code>\markdownSetupSnippet</code>	<i>88, 88</i>
<code>\markdownWarning</code>	<i>84</i>
<code>new</code>	<i>6, 223</i>
<code>notes</code>	<i>25, 75</i>
<code>os.execute</code>	<i>85, 241</i>
<code>\PackageError</code>	<i>86, 246</i>
<code>\PackageInfo</code>	<i>86, 246</i>
<code>\PackageWarning</code>	<i>86, 246</i>
<code>parsers</code>	<i>166, 178</i>
<code>parsers.commented_line</code>	<i>168</i>
<code>\pdfshellescape</code>	<i>241</i>
<code>pipeTables</code>	<i>6, 30, 35, 77</i>
<code>preserveTabs</code>	<i>30, 34, 178</i>
<code>print</code>	<i>242, 243</i>
<code>rawAttribute</code>	<i>31, 31, 82</i>
<code>reader</code>	<i>7, 106, 166, 177, 197</i>
<code>reader-&gt;add_special_character</code>	<i>7, 8, 192</i>
<code>reader-&gt;create_parser</code>	<i>179</i>
<code>reader-&gt;finalize_grammar</code>	<i>189</i>
<code>reader-&gt;insert_pattern</code>	<i>7, 8, 189, 190, 194</i>
<code>reader-&gt;normalize_tag</code>	<i>178</i>
<code>reader-&gt;options</code>	<i>178</i>
<code>reader-&gt;parser_functions</code>	<i>179</i>
<code>reader-&gt;parser_functions.name</code>	<i>179</i>
<code>reader-&gt;parsers</code>	<i>178, 178</i>
<code>reader-&gt;update_rule</code>	<i>189, 192, 194</i>
<code>reader-&gt;writer</code>	<i>178</i>
<code>reader.new</code>	<i>177, 177</i>
<code>relativeReferences</code>	<i>31</i>
<code>\setupmarkdown</code>	<i>104, 104</i>
<code>\shellescape</code>	<i>241</i>
<code>shiftHeadings</code>	<i>6, 32</i>
<code>slice</code>	<i>6, 27, 32, 156</i>
<code>smartEllipses</code>	<i>33, 51</i>
<code>\startmarkdown</code>	<i>103, 103, 271</i>
<code>startNumber</code>	<i>33, 61, 62</i>

status.shell_escape	241
\stopmarkdown	103, 103, 271
strikeThrough	33, 81, 269
stripIndent	34, 179
subscripts	34, 81
superscripts	34, 81
syntax	190, 194
tableCaptions	6, 35
taskLists	35, 49, 259
tex.print	242, 243
texComments	36, 179
\textcites	260
tightLists	36, 57, 59, 60, 63, 64, 66, 252
\title	268
underscores	37
\url	4
\usepackage	87, 90
\usetheme	90
util.cache	107, 107
util.cache_verbatim	107
util.encode_json_string	108
util.err	107
util.escaper	110
util.expand_tabs_in_line	108
util.flatten	109
util.intersperse	110
util.lookup_files	108
util.map	110
util.pathname	111
util.rope_last	110
util.rope_to_string	109
util.table_copy	107
util.walk	108, 109, 110
\VerbatimInput	4
walkable_syntax	7, 15, 20, 189, 190, 192, 194
writer	106, 106, 155, 156, 197
writer->active_attributes	162
writer->active_headings	162
writer->block_html_comment	161

writer->block_html_element	161
writer->blockquote	162
writer->bulletitem	160
writer->bulletlist	159
writer->citation	198
writer->citations	198
writer->code	159
writer->codeFence	206
writer->contentblock	202
writer->defer_call	165, 165
writer->definitionlist	204
writer->document	162
writer->ellipsis	157
writer->emphasis	161
writer->escape	158, 158
writer->escape_minimal	158, 158, 198
writer->escape_uri	158, 158
writer->escaped_chars	158, 158
writer->escaped_minimal_strings	158, 158
writer->escaped_uri_chars	158, 158
writer->fancyitem	220
writer->fancylist	219
writer->get_state	165
writer->heading	163
writer->hybrid	198
writer->image	159
writer->inline_html_comment	161
writer->inline_html_tag	161
writer->interblocksep	157
writer->is_writing	156, 156
writer->jekyllData	210
writer->linebreak	157
writer->link	159
writer->nbsp	157
writer->note	207
writer->options	156
writer->ordereditem	160
writer->orderedlist	160
writer->pack	157, 196
writer->paragraph	157
writer->plain	157
writer->rawBlock	216

writer->rawInline	216
writer->set_state	165
writer->slice_begin	156
writer->slice_end	156
writer->space	157
writer->strike_through	218
writer->string	158
writer->strong	162
writer->subscript	219
writer->suffix	157
writer->superscript	218
writer->table	214
writer->thematic_break	157
writer->checkbox	161
writer->uri	158
writer->verbatim	162
writer.new	156, 156
\writestatus	102