

# A Markdown Interpreter for T<sub>E</sub>X

Vít Novotný  
witiko@mail.muni.cz

Version 2.15.4-0-g4cbe4e3  
2022/07/29

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>87</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	88
1.2	Feedback . . . . .	5	3.2	Plain T <sub>E</sub> X Implementation	198
1.3	Acknowledgements . . . . .	6	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	213
<b>2</b>	<b>Interfaces</b>	<b>6</b>	3.4	ConT <sub>E</sub> Xt Implementation	233
2.1	Lua Interface . . . . .	6			
2.2	Plain T <sub>E</sub> X Interface . . . . .	34			
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	72			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	86			
				<b>References</b>	<b>239</b>
				<b>Index</b>	<b>239</b>

## List of Figures

1	A block diagram of the Markdown package . . . . .	7
2	A sequence diagram of typesetting a document using the T <sub>E</sub> X interface . . . . .	31
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	31
4	Various formats of mathematical formulae . . . . .	79
5	The banner of the Markdown package . . . . .	80
6	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	149

## 1 Introduction

The Markdown package<sup>1</sup> converts markdown<sup>2</sup> markup to T<sub>E</sub>X commands. The functionality is provided both as a Lua module and as plain T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and ConT<sub>E</sub>Xt macro packages that can be used to directly typeset T<sub>E</sub>X documents containing markdown markup. Unlike other convertors, 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

---

<sup>1</sup>See <https://ctan.org/pkg/markdown>.

<sup>2</sup>See <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.<sup>3</sup>

```
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<sub>TEX</sub> 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<sub>TEX</sub>  $\geq$  0.72.0 (TeXLive  $\geq$  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 footnote tags to the lower case. Selene Unicode is included in all releases of Lua<sub>TEX</sub> (TeXLive  $\geq$  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.

---

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

```

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

```

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

```

21 local md5 = require("md5")

```

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [1, Section 3.3]. Beside these, we also carry the following third-party Lua libraries:

**api7/luatinyaml** 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<sup>A</sup>T<sub>E</sub>X3 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 3.2].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 3.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.5), 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<sup>A</sup>T<sub>E</sub>X Requirements

The L<sup>A</sup>T<sub>E</sub>X part of the package requires that the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> format is loaded,

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

a TeX engine that extends  $\varepsilon$ -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<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.2.2), and to provide default token renderer prototypes.

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

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

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

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

**catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X 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.

**expl3** A package that enables the expl3 language from the L<sup>A</sup>T<sub>E</sub>X3 kernel in  $\text{T}_{\text{E}}\text{X Live} \leq 2019$ . It is used in the default renderer prototypes for links (see Section ??), YAML metadata (see Section 3.3.4.6), and in the implementation of L<sup>A</sup>T<sub>E</sub>X themes (see Section 3.3.2.1).

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

#### 1.1.4 ConT<sub>E</sub>Xt Prerequisites

The ConT<sub>E</sub>Xt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T<sub>E</sub>X prerequisites (see Section 1.1.2), and the following ConT<sub>E</sub>Xt 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 GitHub<sup>4</sup> 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<sub>E</sub>X-L<sup>A</sup>T<sub>E</sub>X Stack Exchange.<sup>5</sup> community question answering web site under the [markdown](#) tag.

---

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

<sup>5</sup>See <https://tex.stackexchange.com>.

### 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.

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

Funding by the 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}
```

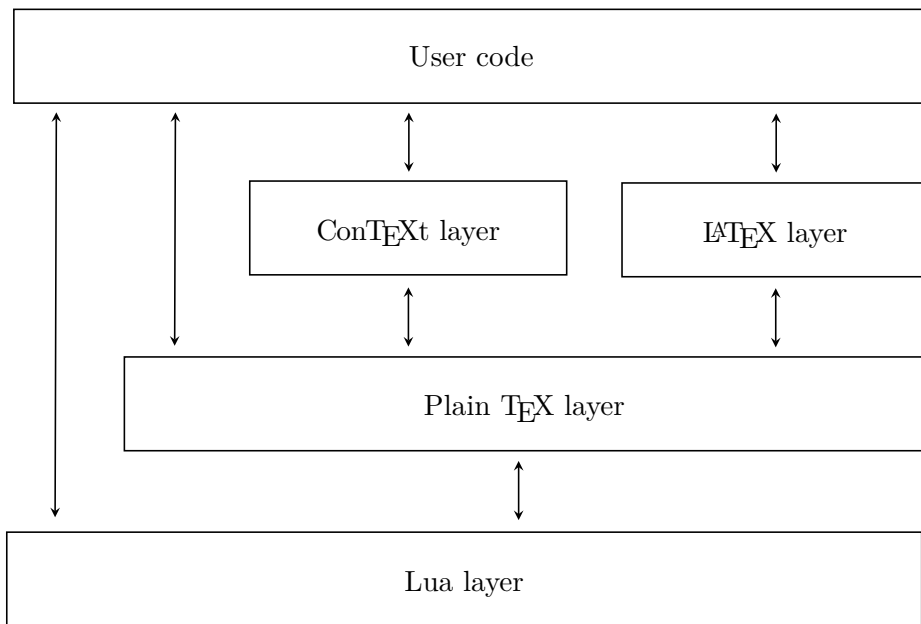


Figure 1: A block diagram of the Markdown package

### 2.1.1 Conversion from Markdown to Plain T<sub>E</sub>X

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

The following example Lua code converts the markdown string `Hello *world*!` to a T<sub>E</sub>X output using the default options and prints the T<sub>E</sub>X output:

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

### 2.1.2 Options

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

```
29 local defaultOptions = {}
```

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

```

30 \ExplSyntaxOn
31 \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.

```

32 \prop_new:N \g_@@_lua_option_types_prop
33 \prop_new:N \g_@@_default_lua_options_prop
34 \seq_new:N \g_@@_option_layers_seq
35 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
36 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
37 \cs_new:Nn
38   \@@_add_lua_option:nnn
39   {
40     \@@_add_option:Vnnn
41     \c_@@_option_layer_lua_tl
42     { #1 }
43     { #2 }
44     { #3 }
45   }
46 \cs_new:Nn
47   \@@_add_option:nnnn
48   {
49     \seq_put_right:cn
50     { g_@@_ #1 _options_seq }
51     { #2 }
52     \prop_put:cnn
53     { g_@@_ #1 _option_types_prop }
54     { #2 }
55     { #3 }
56     \prop_put:cnn
57     { g_@@_default_ #1 _options_prop }
58     { #2 }
59     { #4 }
60     \@@_typecheck_option:n
61     { #2 }
62   }
63 \cs_generate_variant:Nn
64   \@@_add_option:nnnn
65   { Vnnn }
66 \tl_const:Nn \c_@@_option_value_true_tl { true }
67 \tl_const:Nn \c_@@_option_value_false_tl { false }
68 \cs_new:Nn \@@_typecheck_option:n
69   {
70     \@@_get_option_type:nN
71     { #1 }
72     \l_tmpa_tl

```



```

73  \str_case_e:Vn
74  \l_tmpa_tl
75  {
76    { \c_@@_option_type_boolean_tl }
77    {
78      \@@_get_option_value:nN
79      { #1 }
80      \l_tmpa_tl
81      \bool_if:nF
82      {
83        \str_if_eq_p:VV
84        \l_tmpa_tl
85        \c_@@_option_value_true_tl ||
86        \str_if_eq_p:VV
87        \l_tmpa_tl
88        \c_@@_option_value_false_tl
89      }
90    {
91      \msg_error:nnnV
92      { @@ }
93      { failed-typecheck-for-boolean-option }
94      { #1 }
95      \l_tmpa_tl
96    }
97  }
98  }
99  }
100 \msg_new:nnn
101 { @@ }
102 { failed-typecheck-for-boolean-option }
103 {
104   Option~#1~has~value~#2,~
105   but~a~boolean~(true~or~false)~was~expected.
106 }
107 \cs_generate_variant:Nn
108 \str_case_e:nn
109 { Vn }
110 \cs_generate_variant:Nn
111 \msg_error:nnnn
112 { nnnV }
113 \seq_new:N \g_@@_option_types_seq
114 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
115 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
116 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
117 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
118 \tl_const:Nn \c_@@_option_type_number_tl { number }
119 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl

```

```

120 \tl_const:Nn \c_@@_option_type_path_tl { path }
121 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
122 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
123 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
124 \tl_const:Nn \c_@@_option_type_string_tl { string }
125 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
126 \cs_new:Nn
127   \@@_get_option_type:nN
128   {
129     \bool_set_false:N
130       \l_tmpa_bool
131     \seq_map_inline:Nn
132       \g_@@_option_layers_seq
133       {
134         \prop_get:cnNT
135           { g_@@_ ##1 _option_types_prop }
136           { #1 }
137         \l_tmpa_tl
138         {
139           \bool_set_true:N
140             \l_tmpa_bool
141           \seq_map_break:
142         }
143       }
144     \bool_if:nF
145       \l_tmpa_bool
146     {
147       \msg_error:nnn
148         { @@ }
149         { undefined-option }
150         { #1 }
151     }
152     \seq_if_in:NVF
153       \g_@@_option_types_seq
154       \l_tmpa_tl
155     {
156       \msg_error:nnnV
157         { @@ }
158         { unknown-option-type }
159         { #1 }
160       \l_tmpa_tl
161     }
162     \tl_set_eq:NN
163       #2
164       \l_tmpa_tl
165   }
166 \msg_new:nnn

```

```

167 { @@ }
168 { unknown-option-type }
169 {
170   Option~#1~has~unknown~type~#2.
171 }
172 \msg_new:nnn
173 { @@ }
174 { undefined-option }
175 {
176   Option~#1~is~undefined.
177 }
178 \cs_new:Nn
179   \@@_get_default_option_value:nN
180   {
181     \bool_set_false:N
182       \l_tmpa_bool
183     \seq_map_inline:Nn
184       \g_@@_option_layers_seq
185       {
186         \prop_get:cnNT
187           { g_@@_default_ ##1 _options_prop }
188           { #1 }
189           #2
190           {
191             \bool_set_true:N
192               \l_tmpa_bool
193             \seq_map_break:
194           }
195         }
196       \bool_if:nF
197         \l_tmpa_bool
198         {
199           \msg_error:nnn
200             { @@ }
201             { undefined-option }
202             { #1 }
203         }
204     }
205 \cs_new:Nn
206   \@@_get_option_value:nN
207   {
208     \@@_option_tl_to_csname:nN
209       { #1 }
210     \l_tmpa_tl
211     \cs_if_free:cTF
212       { \l_tmpa_tl }
213     {

```

```

214     \@@_get_default_option_value:nN
215     { #1 }
216     #2
217   }
218   {
219     \@@_get_option_type:nN
220     { #1 }
221     \l_tmpa_tl
222     \str_if_eq:NNTF
223     \c_@@_option_type_counter_tl
224     \l_tmpa_tl
225     {
226       \@@_option_tl_to_csname:nN
227       { #1 }
228       \l_tmpa_tl
229       \tl_set:Nx
230       #2
231       { \the \cs:w \l_tmpa_tl \cs_end: }
232     }
233     {
234       \@@_option_tl_to_csname:nN
235       { #1 }
236       \l_tmpa_tl
237       \tl_set:Nv
238       #2
239       { \l_tmpa_tl }
240     }
241   }
242 }
243 \cs_new:Nn \@@_option_tl_to_csname:nN
244 {
245   \tl_set:Nn
246   \l_tmpa_tl
247   { \str_uppercase:n { #1 } }
248   \tl_set:Nx
249   #2
250   {
251     markdownOption
252     \tl_head:f { \l_tmpa_tl }
253     \tl_tail:n { #1 }
254   }
255 }

```

### 2.1.3 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<sub>E</sub>X 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.

```
256 \@@_add_lua_option:nnn
257   { cacheDir }
258   { path }
259   { \markdownOptionOutputDir / _markdown_\jobname }
260 defaultOptions.cacheDir = "."
```

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

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

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

```
261 \@@_add_lua_option:nnn
262   { frozenCacheFileName }
263   { path }
264   { \markdownOptionCacheDir / frozenCache.tex }
265 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

### 2.1.4 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.

```
266 \@@_add_lua_option:nnn
267 { blankBeforeBlockquote }
268 { boolean }
269 { false }

270 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.

```
271 \@@_add_lua_option:nnn
272 { blankBeforeCodeFence }
273 { boolean }
274 { false }

275 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.

```
276 \@@_add_lua_option:nnn
277 { blankBeforeHeading }
278 { boolean }
279 { false }

280 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.

```
281 \@@_add_lua_option:nnn
282 { breakableBlockquotes }
283 { boolean }
284 { false }

285 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.

```
286 \@@_add_lua_option:nnn
287 { citationNbsps }
288 { boolean }
289 { true }

290 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.

```
291 \@@_add_lua_option:nnn
292 { citations }
293 { boolean }
294 { false }

295 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.``
```

```
296 \@@_add_lua_option:nnn  
297 { codeSpans }  
298 { boolean }  
299 { true }  
  
300 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.

```
301 \@@_add_lua_option:nnn  
302 { contentBlocks }  
303 { boolean }  
304 { false }  
  
305 defaultOptions.contentBlocks = false
```



`contentBlocksLanguageMap=<filename>`  
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks. See Section 2.2.3.11 for more information.

```
306 \@@_add_lua_option:nnn
307   { contentBlocksLanguageMap }
308   { path }
309   { markdown-languages.json }
310 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`definitionLists=true, false` default: `false`

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with inline markup

:   Definition 2

        { some code, part of Definition 2 }

Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

```
311 \@@_add_lua_option:nnn
312   { definitionLists }
313   { boolean }
314   { false }
315 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.

```
316 \@@_add_lua_option:nnn
317 { eagerCache }
318 { boolean }
319 { true }
320 defaultOptions.eagerCache = true
```

`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}

```

```

321 \@@_add_lua_option:nnn
322   { expectJekyllData }
323   { boolean }
324   { false }
325 defaultOptions.expectJekyllData = 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.

```
326 \@@_add_lua_option:nnn
327 { fencedCode }
328 { boolean }
329 { false }

330 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.

```
331 \@@_add_lua_option:nnm
332 { finalizeCache }
333 { boolean }
334 { false }

335 defaultOptions.finalizeCache = false
```

`footnotes=true, false` default: false

`true` Enable the Pandoc footnote syntax extension:

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

[^1]: Here is the footnote.

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

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

```
{ some.code }
```

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

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

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

```
336 \@@_add_lua_option:nnn
337 { footnotes }
338 { boolean }
339 { false }
340 defaultOptions.footnotes = false
```

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

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

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

```
341 \@@_add_lua_option:nnn
342 { frozenCacheCounter }
343 { counter }
344 { 0 }
345 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.

```
346 \@@_add_lua_option:nnn
347 { hardLineBreaks }
348 { boolean }
349 { false }
350 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.

```
351 \@@_add_lua_option:nnn
352   { hashEnumerators }
353   { boolean }
354   { false }
355 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.

```
356 \@@_add_lua_option:nnn
357   { headerAttributes }
358   { boolean }
359   { false }
360 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.

```
361 \@@_add_lua_option:nnn
362   { html }
363   { boolean }
364   { false }
365 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.

```
366 \@@_add_lua_option:nnn
367   { hybrid }
368   { boolean }
369   { false }
370 defaultOptions.hybrid = false
```

`inlineFootnotes=true, false` default: `false`

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

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

`false` Disable the Pandoc inline footnote syntax extension.

```
371 \@@_add_lua_option:mn  
372 { inlineFootnotes }  
373 { boolean }  
374 { false }  
  
375 defaultOptions.inlineFootnotes = false
```

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

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

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

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

```
376 \@@_add_lua_option:mn  
377 { jeekyllData }  
378 { boolean }  
379 { false }  
  
380 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.

```
381 \@@_add_lua_option:nnn
382 { pipeTables }
383 { boolean }
384 { false }

385 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.

```
386 \@@_add_lua_option:nnn
387 { preserveTabs }
388 { boolean }
389 { false }

390 defaultOptions.preserveTabs = false
```

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

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

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false`      Disable relative references in autolinks.

```
391 \@@_add_lua_option:nnn
392 { relativeReferences }
393 { boolean }
394 { false }

395 defaultOptions.relativeReferences = false
```

---

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

`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.

```
396 \@@_add_lua_option:nnn
397   { shiftHeadings }
398   { number }
399   { 0 }

400 defaultOptions.shiftHeadings = 0
```

`slice`=*<the beginning and the end of a slice>* default:  $\sim$  \$

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 ( $\sim$ ) selects the beginning of a document.
- The dollar sign (\$) selects the end of a document.
- $\sim$ *<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  $\sim$ *<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  $\sim$ *<identifier>*  $\$$ *<identifier>*, i.e. the entire section with the HTML attribute *#<identifier>* will be selected.

```
401 \@@_add_lua_option:nnn
402   { slice }
403   { slice }
404   {  $\sim$ $ }

405 defaultOptions.slice = " $\sim$  $"
```

`smartEllipses`=`true, false` default: `false`

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

```

406 \@@_add_lua_option:nnn
407   { smartEllipses }
408   { boolean }
409   { false }

410 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` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOItem` T<sub>E</sub>X macro.

```

411 \@@_add_lua_option:nnn
412   { startNumber }
413   { boolean }
414   { true }

415 defaultOptions.startNumber = true

```

`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 false:

```

\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.

```

416 \@@_add_lua_option:nnn
417   { stripIndent }
418   { boolean }
419   { false }

420 defaultOptions.stripIndent = 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.

```
421 \@@_add_lua_option:nnn
422 { tableCaptions }
423 { boolean }
424 { false }
425 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.

```
426 \@@_add_lua_option:nnn
427 { taskLists }
428 { boolean }
429 { false }
430 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.

```
431 \@@_add_lua_option:nnn
432   { texComments }
433   { boolean }
434   { false }
435 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.

```
436 \@@_add_lua_option:nnn
437   { tightLists }
438   { boolean }
439   { true }
```

```
440 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.

```
441 \@@_add_lua_option:nnn
442 { underscores }
443 { boolean }
444 { true }
445 \ExplSyntaxOff
446 defaultOptions.underscores = true
```

### 2.1.5 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.

```
447
448 local HELP_STRING = [[
449 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
450 where OPTIONS are documented in the Lua interface section of the
451 technical Markdown package documentation.
```

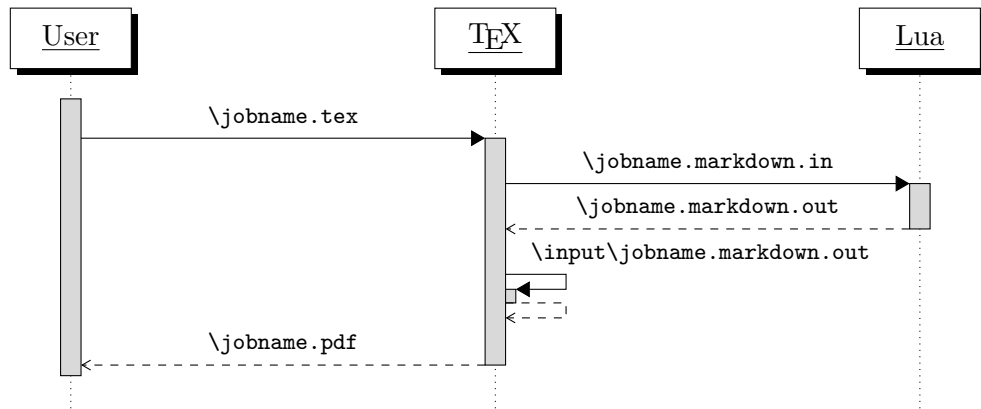


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface

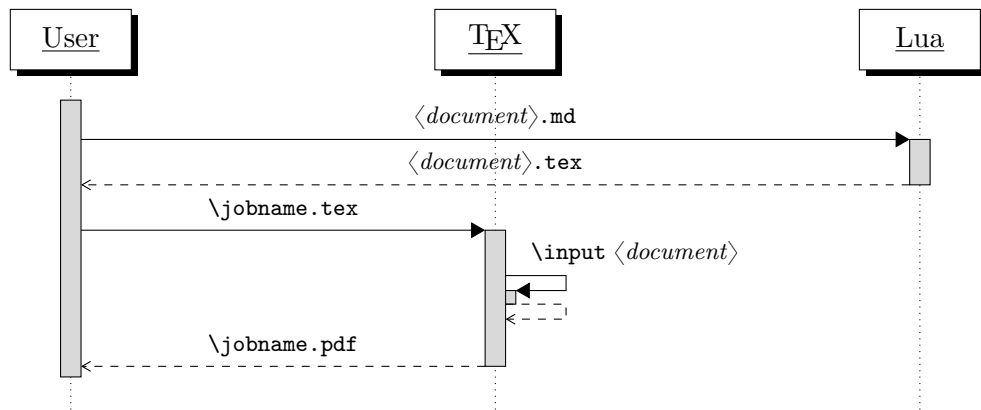


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

452
453 When OUTPUT_FILE is unspecified, the result of the conversion will be
454 written to the standard output. When INPUT_FILE is also unspecified, the
455 result of the conversion will be read from the standard input.
456
457 Report bugs to: witiko@mail.muni.cz
458 Markdown package home page: <https://github.com/witiko/markdown>]]
459
460 local VERSION_STRING = [[
461 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
462
463 Copyright (C) ]] .. table.concat(metadata.copyright,
464                                     "\nCopyright (C) ") .. [[
465
466 License: ]] .. metadata.license
467
468 local function warn(s)
469     io.stderr:write("Warning: " .. s .. "\n") end
470
471 local function error(s)
472     io.stderr:write("Error: " .. s .. "\n")
473     os.exit(1) end
474
475 local process_options = true
476 local options = {}
477 local input_filename
478 local output_filename
479 for i = 1, #arg do
480     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.

```

481         if arg[i] == "--" then
482             process_options = false
483             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 2.1.2.

```

484         elseif arg[i]:match("=") then
485             local key, value = arg[i]:match("(.-)=(.*)")

```

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

```

486             local default_type = type(defaultOptions[key])
487             if default_type == "boolean" then

```



```

488     options[key] = (value == "true")
489 elseif default_type == "number" then
490     options[key] = tonumber(value)
491 else
492     if default_type ~= "string" then
493         if default_type == "nil" then
494             warn('Option "' .. key .. '" not recognized.')
495         else
496             warn('Option "' .. key .. '" type not recognized, please file ' ..
497                 'a report to the package maintainer.')
498         end
499         warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
500             key .. '" as a string.')
501     end
502     options[key] = value
503 end
504 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.

```

505     elseif arg[i] == "--help" or arg[i] == "-h" then
506         print(HELP_STRING)
507         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.

```

508     elseif arg[i] == "--version" or arg[i] == "-v" then
509         print(VERSION_STRING)
510         os.exit()
511     end
512 end

```

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

```

513 if input_filename == nil then
514     input_filename = arg[i]

```

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

```

515 elseif output_filename == nil then
516     output_filename = arg[i]
517 else
518     error('Unexpected argument: "' .. arg[i] .. "'.')
519 end

```

```
520  ::continue::
521 end
```

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

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

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

```
\input hello
```

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

The plain T<sub>E</sub>X interface provides macros for the typesetting of markdown input from within plain T<sub>E</sub>X, for setting the Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain T<sub>E</sub>X and for changing the way markdown the tokens are rendered.

```
522 \def\markdownLastModified{((LASTMODIFIED))}%
523 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
524 \let\markdownBegin\relax
525 \let\markdownEnd\relax
```

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

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the

input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [4, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

The following example plain T<sub>E</sub>X code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

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

The `\markdownInput` macro accepts a single parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X.

```
526 \let\markdownInput\relax
```

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

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

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

## 2.2.2 Options

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

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

```
527 \ExplSyntaxOn
528 \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.

```
529 \prop_new:N \g_@@_plain_tex_option_types_prop
530 \prop_new:N \g_@@_default_plain_tex_options_prop
531 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
532 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
533 \cs_new:Nn
534   \@@_add_plain_tex_option:nnn
535   {
536     \@@_add_option:Vnnn
537     \c_@@_option_layer_plain_tex_tl
538     { #1 }
539     { #2 }
540     { #3 }
541   }
```

**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`.

```
542 \@@_add_plain_tex_option:nnn
543   { frozenCache }
544   { boolean }
545   { false }
```

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

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `\markdownOptionFinalizeCache` option.

4. Typeset the plain T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain T<sub>E</sub>X in T<sub>E</sub>X engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

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

```
546 \@@_add_plain_tex_option:nnn
547   { helperScriptFileName }
548   { path }
549   { \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.

```
550 \str_new:N
551   \g_luabridge_helper_script_filename_str
552 \tl_gset:Nn
553   \g_luabridge_helper_script_filename_str
554   { \markdownOptionHelperScriptFileName }
```

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

```
555 \@@_add_plain_tex_option:nnn
556   { inputTempFileName }
557   { path }
558   { \jobname.markdown.in }
```

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

```
559 \@@_add_plain_tex_option:nnn
560   { outputTempFileName }
561   { path }
562   { \jobname.markdown.out }
```

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

```
563 \str_new:N
564 \g_luabridge_standard_output_filename_str
565 \tl_gset:Nn
566 \g_luabridge_standard_output_filename_str
567 { \markdownOptionOutputTempFileName }
```

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

```
568 \@@_add_plain_tex_option:nnn
569 { errorTempFileName }
570 { path }
571 { \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.

```
572 \str_new:N
573 \g_luabridge_error_output_filename_str
574 \tl_gset:Nn
575 \g_luabridge_error_output_filename_str
576 { \markdownOptionErrorTempFileName }
```

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

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

```
577 \@@_add_plain_tex_option:nnn
578 { outputDir }
579 { path }
580 { . }
```

The `\markdownOptionOutputDir` macro is also used to set the `\g_luabridge_output_dirname_str` macro from the `lt3luabridge` package.

```
581 \str_new:N
582 \g_luabridge_output_dirname_str
```

```

583 \tl_gset:Nn
584 \g_luabridge_output_dirname_str
585 { \markdownOptionOutputDir }

```

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

Furthermore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.2), which are not processed by the plain TeX implementation, only passed along to Lua.

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

```

586 \cs_new:Nn \@@_plain_tex_define_option_commands:
587 {
588   \seq_map_inline:Nn
589     \g_@@_option_layers_seq
590     {
591       \seq_map_inline:cn
592         { g_@@_ ##1 _options_seq }
593         {
594           \@@_plain_tex_define_option_command:n
595             { ###1 }
596         }
597     }
598 }
599 \cs_new:Nn \@@_plain_tex_define_option_command:n
600 {
601   \@@_get_default_option_value:nN
602     { #1 }
603     \l_tmpa_tl
604   \@@_set_option_value:nV
605     { #1 }
606     \l_tmpa_tl
607 }
608 \cs_new:Nn
609   \@@_set_option_value:nn
610   {
611     \@@_define_option:n
612       { #1 }
613     \@@_get_option_type:nN
614       { #1 }
615       \l_tmpa_tl
616     \str_if_eq:NNTF
617       \c_@@_option_type_counter_tl
618       \l_tmpa_tl
619     {
620       \@@_option_tl_to_csname:nN

```

```

621         { #1 }
622         \l_tmpa_tl
623         \int_gset:cn
624         { \l_tmpa_tl }
625         { #2 }
626     }
627     {
628         \@@_option_tl_to_csname:nN
629         { #1 }
630         \l_tmpa_tl
631         \cs_set:cpn
632         { \l_tmpa_tl }
633         { #2 }
634     }
635 }
636 \cs_generate_variant:Nn
637 \@@_set_option_value:nn
638 { nV }
639 \cs_new:Nn
640 \@@_define_option:n
641 {
642     \@@_option_tl_to_csname:nN
643     { #1 }
644     \l_tmpa_tl
645     \cs_if_free:cT
646     { \l_tmpa_tl }
647     {
648         \@@_get_option_type:nN
649         { #1 }
650         \l_tmpb_tl
651         \str_if_eq:NNT
652         \c_@@_option_type_counter_tl
653         \l_tmpb_tl
654         {
655             \@@_option_tl_to_csname:nN
656             { #1 }
657             \l_tmpa_tl
658             \int_new:c
659             { \l_tmpa_tl }
660         }
661     }
662 }
663 \@@_plain_tex_define_option_commands:

```

**2.2.2.3 Miscellaneous Options** The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when



buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [5] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

664 \seq_put_right:Nn
665   \g_@@_plain_tex_options_seq
666   { stripPercentSigns }
667 \prop_put:Nnn
668   \g_@@_plain_tex_option_types_prop
669   { stripPercentSigns }
670   { boolean }
671 \prop_put:Nnx
672   \g_@@_default_plain_tex_options_prop
673   { stripPercentSigns }
674   { false }
675 \ExplSyntaxOff

```

### 2.2.3 Token Renderers

The following T<sub>E</sub>X 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.

```

676 \ExplSyntaxOn
677 \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.

```

678 \prop_new:N \g_@@_renderer_arities_prop
679 \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.

```

680 \def\markdownRendererTickedBox{%
681   \markdownRendererTickedBoxPrototype}%
682 \ExplSyntaxOn
683 \seq_put_right:Nn
684   \g_@@_renderers_seq
685   { tickedBox }

```

```

686 \prop_put:Nnn
687   \g_@@_renderer_arities_prop
688   { tickedBox }
689   { 0 }
690 \ExplSyntaxOff
691 \def\markdownRendererHalfTickedBox{%
692   \markdownRendererHalfTickedBoxPrototype}%
693 \ExplSyntaxOn
694 \seq_put_right:Nn
695   \g_@@_renderers_seq
696   { halfTickedBox }
697 \prop_put:Nnn
698   \g_@@_renderer_arities_prop
699   { halfTickedBox }
700   { 0 }
701 \ExplSyntaxOff
702 \def\markdownRendererUntickedBox{%
703   \markdownRendererUntickedBoxPrototype}%
704 \ExplSyntaxOn
705 \seq_put_right:Nn
706   \g_@@_renderers_seq
707   { untickedBox }
708 \prop_put:Nnn
709   \g_@@_renderer_arities_prop
710   { untickedBox }
711   { 0 }
712 \ExplSyntaxOff

```

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

A  $\text{T}_{\text{E}}\text{X}$  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.

```

713 \def\markdownRendererDocumentBegin{%
714   \markdownRendererDocumentBeginPrototype}%
715 \ExplSyntaxOn
716 \seq_put_right:Nn
717   \g_@@_renderers_seq
718   { documentBegin }
719 \prop_put:Nnn
720   \g_@@_renderer_arities_prop
721   { documentBegin }
722   { 0 }

```

```

723 \ExplSyntaxOff
724 \def\markdownRendererDocumentEnd{%
725   \markdownRendererDocumentEndPrototype}%
726 \ExplSyntaxOn
727 \seq_put_right:Nn
728   \g_@@_renderers_seq
729   { documentEnd }
730 \prop_put:Nnn
731   \g_@@_renderer_arities_prop
732   { documentEnd }
733   { 0 }
734 \ExplSyntaxOff

```

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

```

735 \def\markdownRendererInterblockSeparator{%
736   \markdownRendererInterblockSeparatorPrototype}%
737 \ExplSyntaxOn
738 \seq_put_right:Nn
739   \g_@@_renderers_seq
740   { interblockSeparator }
741 \prop_put:Nnn
742   \g_@@_renderer_arities_prop
743   { interblockSeparator }
744   { 0 }
745 \ExplSyntaxOff

```

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

```

746 \def\markdownRendererLineBreak{%
747   \markdownRendererLineBreakPrototype}%
748 \ExplSyntaxOn
749 \seq_put_right:Nn
750   \g_@@_renderers_seq
751   { lineBreak }
752 \prop_put:Nnn
753   \g_@@_renderer_arities_prop
754   { lineBreak }
755   { 0 }
756 \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.

```

757 \def\markdownRendererEllipsis{%
758   \markdownRendererEllipsisPrototype}%
759 \ExplSyntaxOn
760 \seq_put_right:Nn
761   \g_@@_renderers_seq
762   { ellipsis }
763 \prop_put:Nnn
764   \g_@@_renderer_arities_prop
765   { ellipsis }
766   { 0 }
767 \ExplSyntaxOff

```

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

```

768 \def\markdownRendererNbsp{%
769   \markdownRendererNbspPrototype}%
770 \ExplSyntaxOn
771 \seq_put_right:Nn
772   \g_@@_renderers_seq
773   { nbsp }
774 \prop_put:Nnn
775   \g_@@_renderer_arities_prop
776   { nbsp }
777   { 0 }
778 \ExplSyntaxOff

```

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

```

779 \def\markdownRendererLeftBrace{%
780   \markdownRendererLeftBracePrototype}%
781 \ExplSyntaxOn
782 \seq_put_right:Nn
783   \g_@@_renderers_seq
784   { leftBrace }
785 \prop_put:Nnn
786   \g_@@_renderer_arities_prop
787   { leftBrace }
788   { 0 }
789 \ExplSyntaxOff
790 \def\markdownRendererRightBrace{%

```

```

791 \markdownRendererRightBracePrototype}%
792 \ExplSyntaxOn
793 \seq_put_right:Nn
794 \g_@@_renderers_seq
795 { rightBrace }
796 \prop_put:Nnn
797 \g_@@_renderer_arities_prop
798 { rightBrace }
799 { 0 }
800 \ExplSyntaxOff
801 \def\markdownRendererDollarSign{%
802 \markdownRendererDollarSignPrototype}%
803 \ExplSyntaxOn
804 \seq_put_right:Nn
805 \g_@@_renderers_seq
806 { dollarSign }
807 \prop_put:Nnn
808 \g_@@_renderer_arities_prop
809 { dollarSign }
810 { 0 }
811 \ExplSyntaxOff
812 \def\markdownRendererPercentSign{%
813 \markdownRendererPercentSignPrototype}%
814 \ExplSyntaxOn
815 \seq_put_right:Nn
816 \g_@@_renderers_seq
817 { percentSign }
818 \prop_put:Nnn
819 \g_@@_renderer_arities_prop
820 { percentSign }
821 { 0 }
822 \ExplSyntaxOff
823 \def\markdownRendererAmpersand{%
824 \markdownRendererAmpersandPrototype}%
825 \ExplSyntaxOn
826 \seq_put_right:Nn
827 \g_@@_renderers_seq
828 { ampersand }
829 \prop_put:Nnn
830 \g_@@_renderer_arities_prop
831 { ampersand }
832 { 0 }
833 \ExplSyntaxOff
834 \def\markdownRendererUnderscore{%
835 \markdownRendererUnderscorePrototype}%
836 \ExplSyntaxOn
837 \seq_put_right:Nn

```

```

838 \g_@@_renderers_seq
839 { underscore }
840 \prop_put:Nnn
841 \g_@@_renderer_arities_prop
842 { underscore }
843 { 0 }
844 \ExplSyntaxOff
845 \def\markdownRendererHash{%
846 \markdownRendererHashPrototype}%
847 \ExplSyntaxOn
848 \seq_put_right:Nn
849 \g_@@_renderers_seq
850 { hash }
851 \prop_put:Nnn
852 \g_@@_renderer_arities_prop
853 { hash }
854 { 0 }
855 \ExplSyntaxOff
856 \def\markdownRendererCircumflex{%
857 \markdownRendererCircumflexPrototype}%
858 \ExplSyntaxOn
859 \seq_put_right:Nn
860 \g_@@_renderers_seq
861 { circumflex }
862 \prop_put:Nnn
863 \g_@@_renderer_arities_prop
864 { circumflex }
865 { 0 }
866 \ExplSyntaxOff
867 \def\markdownRendererBackslash{%
868 \markdownRendererBackslashPrototype}%
869 \ExplSyntaxOn
870 \seq_put_right:Nn
871 \g_@@_renderers_seq
872 { backslash }
873 \prop_put:Nnn
874 \g_@@_renderer_arities_prop
875 { backslash }
876 { 0 }
877 \ExplSyntaxOff
878 \def\markdownRendererTilde{%
879 \markdownRendererTildePrototype}%
880 \ExplSyntaxOn
881 \seq_put_right:Nn
882 \g_@@_renderers_seq
883 { tilde }
884 \prop_put:Nnn

```

```

885 \g_@@_renderer_arities_prop
886 { tilde }
887 { 0 }
888 \ExplSyntaxOff
889 \def\markdownRendererPipe{%
890 \markdownRendererPipePrototype}%
891 \ExplSyntaxOn
892 \seq_put_right:Nn
893 \g_@@_renderers_seq
894 { pipe }
895 \prop_put:Nnn
896 \g_@@_renderer_arities_prop
897 { pipe }
898 { 0 }
899 \ExplSyntaxOff

```

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

```

900 \def\markdownRendererCodeSpan{%
901 \markdownRendererCodeSpanPrototype}%
902 \ExplSyntaxOn
903 \seq_put_right:Nn
904 \g_@@_renderers_seq
905 { codeSpan }
906 \prop_put:Nnn
907 \g_@@_renderer_arities_prop
908 { codeSpan }
909 { 1 }
910 \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.

```

911 \def\markdownRendererLink{%
912 \markdownRendererLinkPrototype}%
913 \ExplSyntaxOn
914 \seq_put_right:Nn
915 \g_@@_renderers_seq
916 { link }
917 \prop_put:Nnn
918 \g_@@_renderer_arities_prop
919 { link }
920 { 4 }
921 \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.

```

922 \def\markdownRendererImage{%
923   \markdownRendererImagePrototype}%
924 \ExplSyntaxOn
925 \seq_put_right:Nn
926   \g_@@_renderers_seq
927   { image }
928 \prop_put:Nnn
929   \g_@@_renderer_arities_prop
930   { image }
931   { 4 }
932 \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.

```

933 \def\markdownRendererContentBlock{%
934   \markdownRendererContentBlockPrototype}%
935 \ExplSyntaxOn
936 \seq_put_right:Nn
937   \g_@@_renderers_seq
938   { contentBlock }
939 \prop_put:Nnn
940   \g_@@_renderer_arities_prop
941   { contentBlock }
942   { 4 }
943 \ExplSyntaxOff

```

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

```

944 \def\markdownRendererContentBlockOnlineImage{%
945   \markdownRendererContentBlockOnlineImagePrototype}%
946 \ExplSyntaxOn
947 \seq_put_right:Nn
948   \g_@@_renderers_seq
949   { contentBlockOnlineImage }
950 \prop_put:Nnn
951   \g_@@_renderer_arities_prop
952   { contentBlockOnlineImage }

```



```

953 { 4 }
954 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>7</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

955 \def\markdownRendererContentBlockCode{%
956   \markdownRendererContentBlockCodePrototype}%
957 \ExplSyntaxOn
958 \seq_put_right:Nn
959   \g_@@_renderers_seq
960   { contentBlockCode }
961 \prop_put:Nnn
962   \g_@@_renderer_arities_prop
963   { contentBlockCode }
964   { 5 }
965 \ExplSyntaxOff

```

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

```

966 \def\markdownRendererUlBegin{%
967   \markdownRendererUlBeginPrototype}%
968 \ExplSyntaxOn
969 \seq_put_right:Nn
970   \g_@@_renderers_seq
971   { ulBegin }
972 \prop_put:Nnn
973   \g_@@_renderer_arities_prop
974   { ulBegin }
975   { 0 }

```

---

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

976 \ExplSyntaxOff

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

```
977 \def\markdownRendererUlBeginTight{%
978   \markdownRendererUlBeginTightPrototype}%
979 \ExplSyntaxOn
980 \seq_put_right:Nn
981   \g_@@_renderers_seq
982   { ulBeginTight }
983 \prop_put:Nnn
984   \g_@@_renderer_arities_prop
985   { ulBeginTight }
986   { 0 }
987 \ExplSyntaxOff
```

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

```
988 \def\markdownRendererUlItem{%
989   \markdownRendererUlItemPrototype}%
990 \ExplSyntaxOn
991 \seq_put_right:Nn
992   \g_@@_renderers_seq
993   { ulItem }
994 \prop_put:Nnn
995   \g_@@_renderer_arities_prop
996   { ulItem }
997   { 0 }
998 \ExplSyntaxOff
```

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

```
999 \def\markdownRendererUlItemEnd{%
1000   \markdownRendererUlItemEndPrototype}%
1001 \ExplSyntaxOn
1002 \seq_put_right:Nn
1003   \g_@@_renderers_seq
1004   { ulItemEnd }
1005 \prop_put:Nnn
1006   \g_@@_renderer_arities_prop
1007   { ulItemEnd }
1008   { 0 }
1009 \ExplSyntaxOff
```

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

```

1010 \def\markdownRendererUEnd{%
1011   \markdownRendererUEndPrototype}%
1012 \ExplSyntaxOn
1013 \seq_put_right:Nn
1014   \g_@@_renderers_seq
1015   { ulEnd }
1016 \prop_put:Nnn
1017   \g_@@_renderer_arities_prop
1018   { ulEnd }
1019   { 0 }
1020 \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 `false`. The macro receives no arguments.

```

1021 \def\markdownRendererUEndTight{%
1022   \markdownRendererUEndTightPrototype}%
1023 \ExplSyntaxOn
1024 \seq_put_right:Nn
1025   \g_@@_renderers_seq
1026   { ulEndTight }
1027 \prop_put:Nnn
1028   \g_@@_renderer_arities_prop
1029   { ulEndTight }
1030   { 0 }
1031 \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). The macro receives no arguments.

```

1032 \def\markdownRendererOlBegin{%
1033   \markdownRendererOlBeginPrototype}%
1034 \ExplSyntaxOn
1035 \seq_put_right:Nn
1036   \g_@@_renderers_seq
1037   { olBegin }
1038 \prop_put:Nnn
1039   \g_@@_renderer_arities_prop
1040   { olBegin }
1041   { 0 }
1042 \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 `false`. The macro receives no arguments.

```

1043 \def\markdownRendererOlBeginTight{%
1044   \markdownRendererOlBeginTightPrototype}%
1045 \ExplSyntaxOn
1046 \seq_put_right:Nn
1047   \g_@@_renderers_seq
1048   { olBeginTight }
1049 \prop_put:Nnn
1050   \g_@@_renderer_arities_prop
1051   { olBeginTight }
1052   { 0 }
1053 \ExplSyntaxOff

```

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

```

1054 \def\markdownRendererOlItem{%
1055   \markdownRendererOlItemPrototype}%
1056 \ExplSyntaxOn
1057 \seq_put_right:Nn
1058   \g_@@_renderers_seq
1059   { olItem }
1060 \prop_put:Nnn
1061   \g_@@_renderer_arities_prop
1062   { olItem }
1063   { 0 }
1064 \ExplSyntaxOff

```

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

```

1065 \def\markdownRendererOlItemEnd{%
1066   \markdownRendererOlItemEndPrototype}%
1067 \ExplSyntaxOn
1068 \seq_put_right:Nn
1069   \g_@@_renderers_seq
1070   { olItemEnd }
1071 \prop_put:Nnn
1072   \g_@@_renderer_arities_prop
1073   { olItemEnd }
1074   { 0 }
1075 \ExplSyntaxOff

```

The `\markdownRendererOlItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is

enabled. The macro receives a single numeric argument that corresponds to the item number.

```
1076 \def\markdownRendererO1ItemWithNumber{%
1077   \markdownRendererO1ItemWithNumberPrototype}%
1078 \ExplSyntaxOn
1079 \seq_put_right:Nn
1080   \g_@@_renderers_seq
1081   { olItemWithNumber }
1082 \prop_put:Nnn
1083   \g_@@_renderer_arities_prop
1084   { olItemWithNumber }
1085   { 1 }
1086 \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). The macro receives no arguments.

```
1087 \def\markdownRendererO1End{%
1088   \markdownRendererO1EndPrototype}%
1089 \ExplSyntaxOn
1090 \seq_put_right:Nn
1091   \g_@@_renderers_seq
1092   { olEnd }
1093 \prop_put:Nnn
1094   \g_@@_renderer_arities_prop
1095   { olEnd }
1096   { 0 }
1097 \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 `false`. The macro receives no arguments.

```
1098 \def\markdownRendererO1EndTight{%
1099   \markdownRendererO1EndTightPrototype}%
1100 \ExplSyntaxOn
1101 \seq_put_right:Nn
1102   \g_@@_renderers_seq
1103   { olEndTight }
1104 \prop_put:Nnn
1105   \g_@@_renderer_arities_prop
1106   { olEndTight }
1107   { 0 }
1108 \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.

```
1109 \def\markdownRendererDlBegin{%
1110   \markdownRendererDlBeginPrototype}%
1111 \ExplSyntaxOn
1112 \seq_put_right:Nn
1113   \g_@@_renderers_seq
1114   { dlBegin }
1115 \prop_put:Nnn
1116   \g_@@_renderer_arities_prop
1117   { dlBegin }
1118   { 0 }
1119 \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 `false`. The macro receives no arguments.

```
1120 \def\markdownRendererDlBeginTight{%
1121   \markdownRendererDlBeginTightPrototype}%
1122 \ExplSyntaxOn
1123 \seq_put_right:Nn
1124   \g_@@_renderers_seq
1125   { dlBeginTight }
1126 \prop_put:Nnn
1127   \g_@@_renderer_arities_prop
1128   { dlBeginTight }
1129   { 0 }
1130 \ExplSyntaxOff
```

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

```
1131 \def\markdownRendererDlItem{%
1132   \markdownRendererDlItemPrototype}%
1133 \ExplSyntaxOn
1134 \seq_put_right:Nn
1135   \g_@@_renderers_seq
1136   { dlItem }
1137 \prop_put:Nnn
1138   \g_@@_renderer_arities_prop
1139   { dlItem }
1140   { 1 }
1141 \ExplSyntaxOff
```

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

```
1142 \def\markdownRendererDlItemEnd{%
1143   \markdownRendererDlItemEndPrototype}%
1144 \ExplSyntaxOn
1145 \seq_put_right:Nn
1146   \g_@@_renderers_seq
1147   { dlItemEnd }
1148 \prop_put:Nnn
1149   \g_@@_renderer_arities_prop
1150   { dlItemEnd }
1151   { 0 }
1152 \ExplSyntaxOff
```

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

```
1153 \def\markdownRendererDlDefinitionBegin{%
1154   \markdownRendererDlDefinitionBeginPrototype}%
1155 \ExplSyntaxOn
1156 \seq_put_right:Nn
1157   \g_@@_renderers_seq
1158   { dlDefinitionBegin }
1159 \prop_put:Nnn
1160   \g_@@_renderer_arities_prop
1161   { dlDefinitionBegin }
1162   { 0 }
1163 \ExplSyntaxOff
```

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

```
1164 \def\markdownRendererDlDefinitionEnd{%
1165   \markdownRendererDlDefinitionEndPrototype}%
1166 \ExplSyntaxOn
1167 \seq_put_right:Nn
1168   \g_@@_renderers_seq
1169   { dlDefinitionEnd }
1170 \prop_put:Nnn
1171   \g_@@_renderer_arities_prop
1172   { dlDefinitionEnd }
1173   { 0 }
1174 \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.

```
1175 \def\markdownRendererDlEnd{%
```

```

1176 \markdownRendererDlEndPrototype}%
1177 \ExplSyntaxOn
1178 \seq_put_right:Nn
1179 \g_@@_renderers_seq
1180 { dlEnd }
1181 \prop_put:Nnn
1182 \g_@@_renderer_arities_prop
1183 { dlEnd }
1184 { 0 }
1185 \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 `false`. The macro receives no arguments.

```

1186 \def\markdownRendererDlEndTight{%
1187 \markdownRendererDlEndTightPrototype}%
1188 \ExplSyntaxOn
1189 \seq_put_right:Nn
1190 \g_@@_renderers_seq
1191 { dlEndTight }
1192 \prop_put:Nnn
1193 \g_@@_renderer_arities_prop
1194 { dlEndTight }
1195 { 0 }
1196 \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.

```

1197 \def\markdownRendererEmphasis{%
1198 \markdownRendererEmphasisPrototype}%
1199 \ExplSyntaxOn
1200 \seq_put_right:Nn
1201 \g_@@_renderers_seq
1202 { emphasis }
1203 \prop_put:Nnn
1204 \g_@@_renderer_arities_prop
1205 { emphasis }
1206 { 1 }
1207 \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.



```

1208 \def\markdownRendererStrongEmphasis{%
1209   \markdownRendererStrongEmphasisPrototype}%
1210 \ExplSyntaxOn
1211 \seq_put_right:Nn
1212   \g_@@_renderers_seq
1213   { strongEmphasis }
1214 \prop_put:Nnn
1215   \g_@@_renderer_arities_prop
1216   { strongEmphasis }
1217   { 1 }
1218 \ExplSyntaxOff

```

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

```

1219 \def\markdownRendererBlockQuoteBegin{%
1220   \markdownRendererBlockQuoteBeginPrototype}%
1221 \ExplSyntaxOn
1222 \seq_put_right:Nn
1223   \g_@@_renderers_seq
1224   { blockQuoteBegin }
1225 \prop_put:Nnn
1226   \g_@@_renderer_arities_prop
1227   { blockQuoteBegin }
1228   { 0 }
1229 \ExplSyntaxOff

```

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

```

1230 \def\markdownRendererBlockQuoteEnd{%
1231   \markdownRendererBlockQuoteEndPrototype}%
1232 \ExplSyntaxOn
1233 \seq_put_right:Nn
1234   \g_@@_renderers_seq
1235   { blockQuoteEnd }
1236 \prop_put:Nnn
1237   \g_@@_renderer_arities_prop
1238   { blockQuoteEnd }
1239   { 0 }
1240 \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.

```

1241 \def\markdownRendererInputVerbatim{%
1242   \markdownRendererInputVerbatimPrototype}%

```

```

1243 \ExplSyntaxOn
1244 \seq_put_right:Nn
1245   \g_@@_renderers_seq
1246   { inputVerbatim }
1247 \prop_put:Nnn
1248   \g_@@_renderer_arities_prop
1249   { inputVerbatim }
1250   { 1 }
1251 \ExplSyntaxOff

```

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

```

1252 \def\markdownRendererInputFencedCode{%
1253   \markdownRendererInputFencedCodePrototype}%
1254 \ExplSyntaxOn
1255 \seq_put_right:Nn
1256   \g_@@_renderers_seq
1257   { inputFencedCode }
1258 \prop_put:Nnn
1259   \g_@@_renderer_arities_prop
1260   { inputFencedCode }
1261   { 2 }
1262 \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.

```

1263 \def\markdownRendererJekyllDataBegin{%
1264   \markdownRendererJekyllDataBeginPrototype}%
1265 \ExplSyntaxOn
1266 \seq_put_right:Nn
1267   \g_@@_renderers_seq
1268   { jekyllDataBegin }
1269 \prop_put:Nnn
1270   \g_@@_renderer_arities_prop
1271   { jekyllDataBegin }
1272   { 0 }
1273 \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.

```

1274 \def\markdownRendererJekyllDataEnd{%

```

```

1275 \markdownRendererJekyllDataEndPrototype}%
1276 \ExplSyntaxOn
1277 \seq_put_right:Nn
1278 \g_@@_renderers_seq
1279 { jekyllDataEnd }
1280 \prop_put:Nnn
1281 \g_@@_renderer_arities_prop
1282 { jekyllDataEnd }
1283 { 0 }
1284 \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.

```

1285 \def\markdownRendererJekyllDataMappingBegin{%
1286 \markdownRendererJekyllDataMappingBeginPrototype}%
1287 \ExplSyntaxOn
1288 \seq_put_right:Nn
1289 \g_@@_renderers_seq
1290 { jekyllDataMappingBegin }
1291 \prop_put:Nnn
1292 \g_@@_renderer_arities_prop
1293 { jekyllDataMappingBegin }
1294 { 2 }
1295 \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.

```

1296 \def\markdownRendererJekyllDataMappingEnd{%
1297 \markdownRendererJekyllDataMappingEndPrototype}%
1298 \ExplSyntaxOn
1299 \seq_put_right:Nn
1300 \g_@@_renderers_seq
1301 { jekyllDataMappingEnd }
1302 \prop_put:Nnn
1303 \g_@@_renderer_arities_prop
1304 { jekyllDataMappingEnd }
1305 { 0 }
1306 \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.

```
1307 \def\markdownRendererJekyllDataSequenceBegin{%
1308   \markdownRendererJekyllDataSequenceBeginPrototype}%
1309 \ExplSyntaxOn
1310 \seq_put_right:Nn
1311   \g_@@_renderers_seq
1312   { jekyllDataSequenceBegin }
1313 \prop_put:Nnn
1314   \g_@@_renderer_arities_prop
1315   { jekyllDataSequenceBegin }
1316   { 2 }
1317 \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.

```
1318 \def\markdownRendererJekyllDataSequenceEnd{%
1319   \markdownRendererJekyllDataSequenceEndPrototype}%
1320 \ExplSyntaxOn
1321 \seq_put_right:Nn
1322   \g_@@_renderers_seq
1323   { jekyllDataSequenceEnd }
1324 \prop_put:Nnn
1325   \g_@@_renderer_arities_prop
1326   { jekyllDataSequenceEnd }
1327   { 0 }
1328 \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.

```
1329 \def\markdownRendererJekyllDataBoolean{%
1330   \markdownRendererJekyllDataBooleanPrototype}%
1331 \ExplSyntaxOn
1332 \seq_put_right:Nn
1333   \g_@@_renderers_seq
1334   { jekyllDataBoolean }
1335 \prop_put:Nnn
1336   \g_@@_renderer_arities_prop
1337   { jekyllDataBoolean }
1338   { 2 }
1339 \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.

```

1340 \def\markdownRendererJekyllDataNumber{%
1341   \markdownRendererJekyllDataNumberPrototype}%
1342 \ExplSyntaxOn
1343 \seq_put_right:Nn
1344   \g_@@_renderers_seq
1345   { jekyllDataNumber }
1346 \prop_put:Nnn
1347   \g_@@_renderer_arities_prop
1348   { jekyllDataNumber }
1349   { 2 }
1350 \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.

```

1351 \def\markdownRendererJekyllDataString{%
1352   \markdownRendererJekyllDataStringPrototype}%
1353 \ExplSyntaxOn
1354 \seq_put_right:Nn
1355   \g_@@_renderers_seq
1356   { jekyllDataString }
1357 \prop_put:Nnn
1358   \g_@@_renderer_arities_prop
1359   { jekyllDataString }
1360   { 2 }
1361 \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.

```

1362 \def\markdownRendererJekyllDataEmpty{%
1363   \markdownRendererJekyllDataEmptyPrototype}%
1364 \ExplSyntaxOn
1365 \seq_put_right:Nn
1366   \g_@@_renderers_seq
1367   { jekyllDataEmpty }

```

```

1368 \prop_put:Nnn
1369   \g_@@_renderer_arities_prop
1370   { jekyllDataEmpty }
1371   { 1 }
1372 \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.

```

1373 \def\markdownRendererHeadingOne{%
1374   \markdownRendererHeadingOnePrototype}%
1375 \ExplSyntaxOn
1376 \seq_put_right:Nn
1377   \g_@@_renderers_seq
1378   { headingOne }
1379 \prop_put:Nnn
1380   \g_@@_renderer_arities_prop
1381   { headingOne }
1382   { 1 }
1383 \ExplSyntaxOff

```

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

```

1384 \def\markdownRendererHeadingTwo{%
1385   \markdownRendererHeadingTwoPrototype}%
1386 \ExplSyntaxOn
1387 \seq_put_right:Nn
1388   \g_@@_renderers_seq
1389   { headingTwo }
1390 \prop_put:Nnn
1391   \g_@@_renderer_arities_prop
1392   { headingTwo }
1393   { 1 }
1394 \ExplSyntaxOff

```

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

```

1395 \def\markdownRendererHeadingThree{%
1396   \markdownRendererHeadingThreePrototype}%
1397 \ExplSyntaxOn
1398 \seq_put_right:Nn
1399   \g_@@_renderers_seq
1400   { headingThree }
1401 \prop_put:Nnn
1402   \g_@@_renderer_arities_prop
1403   { headingThree }

```

```
1404 { 1 }
1405 \ExplSyntaxOff
```

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

```
1406 \def\markdownRendererHeadingFour{%
1407   \markdownRendererHeadingFourPrototype}%
1408 \ExplSyntaxOn
1409 \seq_put_right:Nn
1410   \g_@@_renderers_seq
1411   { headingFour }
1412 \prop_put:Nnn
1413   \g_@@_renderer_arities_prop
1414   { headingFour }
1415   { 1 }
1416 \ExplSyntaxOff
```

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

```
1417 \def\markdownRendererHeadingFive{%
1418   \markdownRendererHeadingFivePrototype}%
1419 \ExplSyntaxOn
1420 \seq_put_right:Nn
1421   \g_@@_renderers_seq
1422   { headingFive }
1423 \prop_put:Nnn
1424   \g_@@_renderer_arities_prop
1425   { headingFive }
1426   { 1 }
1427 \ExplSyntaxOff
```

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

```
1428 \def\markdownRendererHeadingSix{%
1429   \markdownRendererHeadingSixPrototype}%
1430 \ExplSyntaxOn
1431 \seq_put_right:Nn
1432   \g_@@_renderers_seq
1433   { headingSix }
1434 \prop_put:Nnn
1435   \g_@@_renderer_arities_prop
1436   { headingSix }
1437   { 1 }
1438 \ExplSyntaxOff
```

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

```

1439 \def\markdownRendererHorizontalRule{%
1440   \markdownRendererHorizontalRulePrototype}%
1441 \ExplSyntaxOn
1442 \seq_put_right:Nn
1443   \g_@@_renderers_seq
1444   { horizontalRule }
1445 \prop_put:Nnn
1446   \g_@@_renderer_arities_prop
1447   { horizontalRule }
1448   { 0 }
1449 \ExplSyntaxOff

```

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

```

1450 \def\markdownRendererFootnote{%
1451   \markdownRendererFootnotePrototype}%
1452 \ExplSyntaxOn
1453 \seq_put_right:Nn
1454   \g_@@_renderers_seq
1455   { footnote }
1456 \prop_put:Nnn
1457   \g_@@_renderer_arities_prop
1458   { footnote }
1459   { 1 }
1460 \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.

```

1461 \def\markdownRendererCite{%
1462   \markdownRendererCitePrototype}%
1463 \ExplSyntaxOn
1464 \seq_put_right:Nn
1465   \g_@@_renderers_seq
1466   { cite }
1467 \prop_put:Nnn
1468   \g_@@_renderer_arities_prop

```



```

1469 { cite }
1470 { 1 }
1471 \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.

```

1472 \def\markdownRendererTextCite{%
1473   \markdownRendererTextCitePrototype}%
1474 \ExplSyntaxOn
1475 \seq_put_right:Nn
1476   \g_@@_renderers_seq
1477   { textCite }
1478 \prop_put:Nnn
1479   \g_@@_renderer_arities_prop
1480   { textCite }
1481   { 1 }
1482 \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.

```

1483 \def\markdownRendererTable{%
1484   \markdownRendererTablePrototype}%
1485 \ExplSyntaxOn
1486 \seq_put_right:Nn
1487   \g_@@_renderers_seq
1488   { table }
1489 \prop_put:Nnn
1490   \g_@@_renderer_arities_prop
1491   { table }
1492   { 3 }
1493 \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.

```

1494 \def\markdownRendererInlineHtmlComment{%
1495   \markdownRendererInlineHtmlCommentPrototype}%
1496 \ExplSyntaxOn
1497 \seq_put_right:Nn
1498   \g_@@_renderers_seq
1499   { inlineHtmlComment }
1500 \prop_put:Nnn
1501   \g_@@_renderer_arities_prop
1502   { inlineHtmlComment }
1503   { 1 }
1504 \ExplSyntaxOff
1505 \def\markdownRendererBlockHtmlCommentBegin{%
1506   \markdownRendererBlockHtmlCommentBeginPrototype}%
1507 \ExplSyntaxOn
1508 \seq_put_right:Nn
1509   \g_@@_renderers_seq
1510   { blockHtmlCommentBegin }
1511 \prop_put:Nnn
1512   \g_@@_renderer_arities_prop
1513   { blockHtmlCommentBegin }
1514   { 0 }
1515 \ExplSyntaxOff
1516 \def\markdownRendererBlockHtmlCommentEnd{%
1517   \markdownRendererBlockHtmlCommentEndPrototype}%
1518 \ExplSyntaxOn
1519 \seq_put_right:Nn
1520   \g_@@_renderers_seq
1521   { blockHtmlCommentEnd }
1522 \prop_put:Nnn
1523   \g_@@_renderer_arities_prop
1524   { blockHtmlCommentEnd }
1525   { 0 }
1526 \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.

```

1527 \def\markdownRendererInlineHtmlTag{%
1528   \markdownRendererInlineHtmlTagPrototype}%
1529 \ExplSyntaxOn
1530 \seq_put_right:Nn
1531   \g_@@_renderers_seq
1532   { inlineHtmlTag }
1533 \prop_put:Nnn
1534   \g_@@_renderer_arities_prop
1535   { inlineHtmlTag }
1536   { 1 }
1537 \ExplSyntaxOff
1538 \def\markdownRendererInputBlockHtmlElement{%
1539   \markdownRendererInputBlockHtmlElementPrototype}%
1540 \ExplSyntaxOn
1541 \seq_put_right:Nn
1542   \g_@@_renderers_seq
1543   { inputBlockHtmlElement }
1544 \prop_put:Nnn
1545   \g_@@_renderer_arities_prop
1546   { inputBlockHtmlElement }
1547   { 1 }
1548 \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  $\langle identifier \rangle$ .

`\markdownRendererAttributeClassName` 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.

```

1549 \def\markdownRendererAttributeIdentifier{%
1550   \markdownRendererAttributeIdentifierPrototype}%
1551 \ExplSyntaxOn
1552 \seq_put_right:Nn

```

```

1553 \g_@@_renderers_seq
1554 { attributeIdentifier }
1555 \prop_put:Nnn
1556 \g_@@_renderer_arities_prop
1557 { attributeIdentifier }
1558 { 1 }
1559 \ExplSyntaxOff
1560 \def\markdownRendererAttributeClassName{%
1561 \markdownRendererAttributeClassNamePrototype}%
1562 \ExplSyntaxOn
1563 \seq_put_right:Nn
1564 \g_@@_renderers_seq
1565 { attributeClassName }
1566 \prop_put:Nnn
1567 \g_@@_renderer_arities_prop
1568 { attributeClassName }
1569 { 1 }
1570 \ExplSyntaxOff
1571 \def\markdownRendererAttributeKeyValue{%
1572 \markdownRendererAttributeKeyValuePrototype}%
1573 \ExplSyntaxOn
1574 \seq_put_right:Nn
1575 \g_@@_renderers_seq
1576 { attributeKeyValue }
1577 \prop_put:Nnn
1578 \g_@@_renderer_arities_prop
1579 { attributeKeyValue }
1580 { 2 }
1581 \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.

```

1582 \def\markdownRendererHeaderAttributeContextBegin{%
1583 \markdownRendererHeaderAttributeContextBeginPrototype}%
1584 \ExplSyntaxOn
1585 \seq_put_right:Nn
1586 \g_@@_renderers_seq
1587 { headerAttributeContextBegin }
1588 \prop_put:Nnn
1589 \g_@@_renderer_arities_prop
1590 { headerAttributeContextBegin }
1591 { 0 }
1592 \ExplSyntaxOff

```

```

1593 \def\markdownRendererHeaderAttributeContextEnd{%
1594   \markdownRendererHeaderAttributeContextEndPrototype}%
1595 \ExplSyntaxOn
1596 \seq_put_right:Nn
1597   \g_@@_renderers_seq
1598   { headerAttributeContextEnd }
1599 \prop_put:Nnn
1600   \g_@@_renderer_arities_prop
1601   { headerAttributeContextEnd }
1602   { 0 }
1603 \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  $\LaTeX$ 3 kernel.

```

1604 \ExplSyntaxOn
1605 \keys_define:nn
1606   { markdown/jekyllData }
1607   { }
1608 \ExplSyntaxOff

```

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

```

1609 \ExplSyntaxOn
1610 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
1611   {
1612     \seq_map_function:NN
1613       \g_@@_renderers_seq
1614       \@@_plaintex_define_renderer_prototype:n
1615       \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
1616       \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1617       \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1618       \let\markdownRendererBlockHtmlCommentEnd=\fi
1619   }
1620 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
1621   {
1622     \@@_renderer_prototype_tl_to_csname:nN
1623       { #1 }
1624       \l_tmpa_tl
1625     \prop_get:NnN
1626       \g_@@_renderer_arities_prop

```

```

1627     { #1 }
1628     \l_tmpb_tl
1629     \@@_plaintex_define_renderer_prototype:cV
1630     { \l_tmpa_tl }
1631     \l_tmpb_tl
1632   }
1633 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
1634 {
1635   \tl_set:Nn
1636     \l_tmpa_tl
1637     { \str_uppercase:n { #1 } }
1638   \tl_set:Nx
1639     #2
1640     {
1641       markdownRenderer
1642       \tl_head:f { \l_tmpa_tl }
1643       \tl_tail:n { #1 }
1644       Prototype
1645     }
1646   }
1647 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
1648 {
1649   \cs_generate_from_arg_count:NNnn
1650     #1
1651     \cs_set:Npn
1652     { #2 }
1653     { }
1654   }
1655 \cs_generate_variant:Nn
1656   \@@_plaintex_define_renderer_prototype:Nn
1657   { cV }
1658 \@@_plaintex_define_renderer_prototypes:
1659 \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 T<sub>E</sub>X special characters to other, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
1660 \let\markdownMakeOther\relax
```

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

```
1661 \let\markdownReadAndConvert\relax
```

```
1662 \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*.

```
1663 \catcode`\|=0\catcode`\=12%
```

```
1664 |gdef|markdownBegin{%
```

```
1665     |markdownReadAndConvert{\markdownEnd}%
```

```
1666                                     {\|markdownEnd}}%
```

```
1667 |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 T<sub>E</sub>X 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 T<sub>E</sub>X 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.

```
1668 \ExplSyntaxOn
```

```
1669 \cs_if_exist:NF
```

```
1670   \markdownMode
```

```
1671   {
```

```
1672     \file_if_exist:nTF
```

```

1673     { lt3luabridge.tex }
1674     {
1675         \cs_new:Npn
1676             \markdownMode
1677             { 3 }
1678     }
1679     {
1680         \cs_if_exist:NTF
1681             \directlua
1682             {
1683                 \cs_new:Npn
1684                     \markdownMode
1685                     { 2 }
1686             }
1687         {
1688             \cs_new:Npn
1689                 \markdownMode
1690                 { 0 }
1691         }
1692     }
1693 }

1694 \int_compare:nF
1695 { \markdownMode = 3 }
1696 {
1697     \int_new:N
1698         \g_luabridge_method_int
1699     \int_gset:Nn
1700         \g_luabridge_method_int
1701         { \markdownMode }
1702 }
1703 \ExplSyntaxOff

```

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

```

1704 \def\markdownLuaRegisterIBCallback#1{\relax}%
1705 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

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

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).



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

```
1706 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
1707 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
1708 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
1709 \input markdown/markdown
```

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

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

where `<options>` are the  $\LaTeX$  interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) 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.

```
1710 \newenvironment{markdown}\relax\relax
1711 \newenvironment{markdown*}[1]\relax\relax
```

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

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

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

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

<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  $\text{T}_{\text{E}}\text{X}$ . Unlike the `\markdownInput` macro provided by the plain  $\text{T}_{\text{E}}\text{X}$  interface, this macro also accepts  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  code showcases the usage of the `\markdownInput` macro:

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

### 2.3.2 Options

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

Except for the `plain` option described in Section 2.3.2.1, and the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  themes described in Section 2.3.2.2, and the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  setup snippets described in Section 2.3.2.3,  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options map directly to the options recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Sections 2.2.3 and 2.2.4).

The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options may be specified when loading the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  package, when using the `markdown*`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
1712 \ExplSyntaxOn
1713 \cs_new:Nn
1714   \@@_setup:n
1715   {
1716     \keys_set:nn
1717       { markdown/latex-options }
1718       { #1 }
1719   }
1720 \let\markdownSetup=\@@_setup:n
1721 \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:

```

1722 \newcommand\markdownSetupSnippet[2]{%
1723   \markdownIfSnippetExists{#1}%
1724   {%
1725     \markdownWarning
1726     {Redefined setup snippet \markdownLaTeXThemeName#1}%
1727     \csname markdownLaTeXSetupSnippet%
1728     \markdownLaTeXThemeName#1\endcsname={#2}%
1729   }{%
1730     \newtoks\next
1731     \next={#2}%
1732     \expandafter\let\csname markdownLaTeXSetupSnippet%
1733     \markdownLaTeXThemeName#1\endcsname=\next
1734   }}%

```

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

```

1735 \newcommand\markdownIfSnippetExists[3]{%
1736   \@ifundefined
1737   {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
1738   {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and  $\LaTeX$  themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of  $\LaTeX$  options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

1739 \ExplSyntaxOn
1740 \seq_new:N \g_@@_latex_options_seq

```

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

```

1741 \prop_new:N \g_@@_latex_option_types_prop
1742 \prop_new:N \g_@@_default_latex_options_prop
1743 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
1744 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
1745 \cs_new:Nn
1746   \@@_add_latex_option:nnn
1747   {
1748     \@@_add_option:Vnnn
1749     \c_@@_option_layer_latex_tl
1750     { #1 }
1751     { #2 }
1752     { #3 }
1753   }

```

**2.3.2.1 No default token renderer prototypes** Default token renderer prototypes require  $\LaTeX$  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  $\TeX$  implementation (see Section 3.2.2) and prevent the soft  $\LaTeX$  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}
```

```
1754 \@_add_latex_option:nnn
1755   { plain }
1756   { boolean }
1757   { false }
1758 \ExplSyntaxOff
```

**2.3.2.2  $\LaTeX$  themes** User-contributed  $\LaTeX$  themes for the Markdown package provide a domain-specific interpretation of some Markdown tokens. Similarly to  $\LaTeX$  packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The  $\LaTeX$  option with key `theme` loads a  $\LaTeX$  package (further referred to as *a theme*) named `markdowntheme<munged theme name>.sty`, where the *munged theme name* is the *theme name* after a substitution of all forward slashes (/) for an underscore (\_), the theme name is a value that 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  $\LaTeX$  package, which provides similar functionality with its `\usetheme` macro [6, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single  $\LaTeX$  document class or for a single  $\LaTeX$  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  $\LaTeX$  packages are identified only by their filenames, not by their pathnames. [7] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a  $\LaTeX$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\LaTeX$  option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until

after the Markdown L<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

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

```
1759 \newif\ifmarkdownLaTeXLoaded
1760 \markdownLaTeXLoadedfalse
1761 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
1762 \ExplSyntaxOn
1763 \tl_new:N \markdownLaTeXThemePackageName
1764 \cs_new:Nn
1765   \@@_set_latex_theme:n
1766   {
1767     \str_if_in:NnF
1768       { #1 }
1769       { / }
1770       {
1771         \markdownError
1772         { Won't load theme with unqualified name #1 }
1773         { Theme names must contain at least one forward slash }
1774       }
1775     \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
1776     \str_replace_all:Nnn
1777       \markdownLaTeXThemePackageName
1778       { / }
1779       { _ }
1780     \edef\markdownLaTeXThemePackageName{
1781       markdowntheme\markdownLaTeXThemePackageName}
1782     \expandafter\markdownLaTeXThemeLoad\expandafter{
1783       \markdownLaTeXThemePackageName}{#1/}
1784   }
1785 \keys_define:nn
1786   { markdown/latex-options }
1787   {
1788     theme .code:n = { \@@_set_latex_theme:n { #1 } },
1789   }
1790 \ExplSyntaxOff
```

The  $\LaTeX$  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  $\LaTeX$ , themes may not be loaded after the beginning of a  $\LaTeX$  document.

```
1791 \ExplSyntaxOn
1792 \@onlypreamble
1793   \@@_set_latex_theme:n
1794 \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;

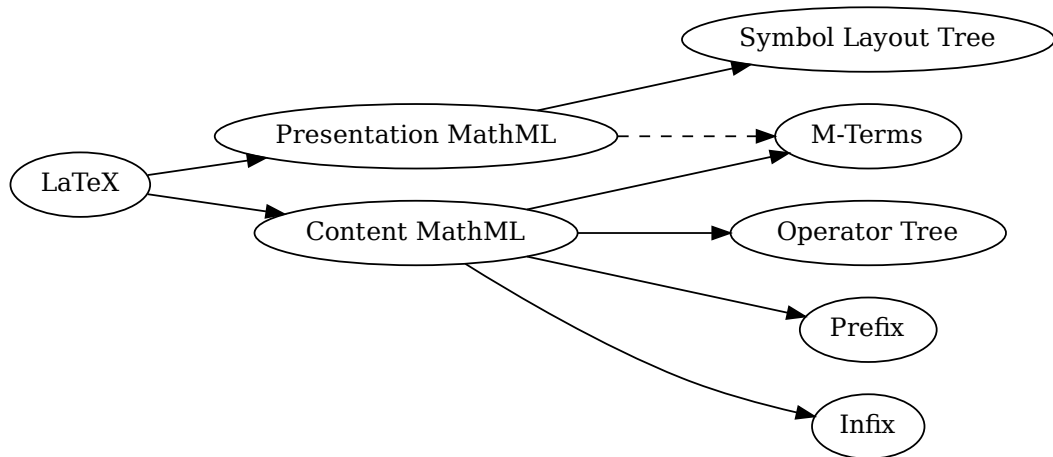
  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
```

```

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 T<sub>E</sub>X option is enabled.

1795 \ProvidesPackage{markdownthemewitiko\_dot}[2021/03/09]%

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

```

\documentclass{article}
\usepackage[theme=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}
! [img] (https://github.com/witiko/markdown/raw/main/markdown.png

```

```

    "The banner of the Markdown package")
\end{markdown}
\end{document}

```

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

```

\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  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain  $\TeX$  option is enabled.

1796 \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 `false`.

```

\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”.

```
1797 \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  $\langle value \rangle$ :

```

1798 \ExplSyntaxOn
1799 \keys_define:nn
1800   { markdown/latex-options }
1801   {
1802     snippet .code:n = {
1803       \markdownIfSnippetExists{#1}
1804       {
1805         \expandafter\markdownSetup\expandafter{
1806           \the\csname markdownLaTeXSetupSnippet
1807             \markdownLaTeXThemeName#1\endcsname}
1808         }{
1809           \markdownError
1810             {Can't~invoke~setup~snippet~#1}
1811             {The~setup~snippet~is~undefined}
1812         }
1813       }
1814     }
1815 \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.

```
1816 \ExplSyntaxOn
1817 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
1818   {
1819     \seq_map_inline:Nn
1820       \g_@@_latex_options_seq
1821       {
1822         \@@_plain_tex_define_option_command:n
1823         { ##1 }
1824       }

```

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

```
1825   \seq_map_inline:Nn
1826     \g_@@_option_layers_seq
1827     {
1828       \seq_map_inline:cn
1829         { g_@@_ ##1 _options_seq }
1830         {
1831           \@@_latex_define_option_keyval:nn
1832           { ##1 }
1833           { #####1 }
1834         }
1835     }
1836 }
1837 \cs_new:Nn \@@_latex_define_option_keyval:nn
1838   {
1839     \prop_get:cnN
1840     { g_@@_ #1 _option_types_prop }
1841     { #2 }
1842     \l_tmpa_tl
1843     \keys_define:nn
1844     { markdown/latex-options }
1845     {

```

```

1846     #2 .code:n = {
1847         \@@_set_option_value:nn
1848         { #2 }
1849         { ##1 }
1850     },
1851 }
1852 \str_if_eq:VVT
1853 \l_tmpa_tl
1854 \c_@@_option_type_boolean_tl
1855 {
1856     \keys_define:nn
1857     { markdown/latex-options }
1858     {
1859         #2 .default:n = { true },
1860     }
1861 }
1862 }
1863 \@@_latex_define_option_commands_and_keyvals:
1864 \ExplSyntaxOff

```

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

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

```

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

```

We hope that other packages will support the `finalizcache` and `frozencache` package options in the future, so that they can become a standard interface for preparing L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```

1865 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
1866 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}

```

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

```

\markdownSetup{
  hybrid,

```

```

smartEllipses,
cacheDir = /tmp,
}

```

**2.3.2.5 Plain T<sub>E</sub>X Markdown Token Renderers** The L<sup>A</sup>T<sub>E</sub>X interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain T<sub>E</sub>X interface (see Section 2.2.3).

```

1867 \ExplSyntaxOn
1868 \cs_new:Nn \@@_latex_define_renderers:
1869 {
1870   \seq_map_function:NN
1871     \g_@@_renderers_seq
1872     \@@_latex_define_renderer:n
1873 }
1874 \cs_new:Nn \@@_latex_define_renderer:n
1875 {
1876   \@@_renderer_tl_to_csname:nN
1877     { #1 }
1878   \l_tmpa_tl
1879   \prop_get:NnN
1880     \g_@@_renderer_arities_prop
1881     { #1 }
1882   \l_tmpb_tl
1883   \@@_latex_define_renderer:ncV
1884     { #1 }
1885     { \l_tmpa_tl }
1886     \l_tmpb_tl
1887 }
1888 \cs_new:Nn \@@_renderer_tl_to_csname:nN
1889 {
1890   \tl_set:Nn
1891     \l_tmpa_tl
1892     { \str_uppercase:n { #1 } }
1893   \tl_set:Nx
1894     #2
1895     {
1896       markdownRenderer
1897       \tl_head:f { \l_tmpa_tl }
1898       \tl_tail:n { #1 }
1899     }
1900 }
1901 \cs_new:Nn \@@_latex_define_renderer:nNn
1902 {
1903   \keys_define:nn

```

```

1904     { markdown/latex-options/renderers }
1905     {
1906         #1 .code:n = {
1907             \cs_generate_from_arg_count:NNnn
1908             #2
1909             \cs_set:Npn
1910             { #3 }
1911             { ##1 }
1912         },
1913     }
1914 }
1915 \cs_generate_variant:Nn
1916   \@@_latex_define_renderer:nNn
1917   { ncV }
1918 \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).

```

1919 \ExplSyntaxOn
1920 \cs_new:Nn \@@_latex_define_renderer_prototypes:
1921   {
1922     \seq_map_function:NN
1923     \g_@@_renderers_seq
1924     \@@_latex_define_renderer_prototype:n
1925   }
1926 \cs_new:Nn \@@_latex_define_renderer_prototype:n
1927   {
1928     \@@_renderer_prototype_tl_to_csname:nN
1929     { #1 }
1930     \l_tmpa_tl
1931     \prop_get:NnN
1932     \g_@@_renderer_arities_prop

```

```

1933     { #1 }
1934     \l_tmpb_tl
1935     \@@_latex_define_renderer_prototype:ncV
1936     { #1 }
1937     { \l_tmpa_tl }
1938     \l_tmpb_tl
1939   }
1940 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
1941 {
1942   \keys_define:nn
1943     { markdown/latex-options/renderer-prototypes }
1944     {
1945       #1 .code:n = {
1946         \cs_generate_from_arg_count:NNnn
1947           #2
1948         \cs_set:Npn
1949           { #3 }
1950           { ##1 }
1951       },
1952     }
1953   }
1954 \cs_generate_variant:Nn
1955   \@@_latex_define_renderer_prototype:nNn
1956   { ncV }
1957 \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. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```

1958 \writestatus{loading}{ConTEXt User Module / markdown}%
1959 \startmodule[markdown]
1960 \unprotect

```

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

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

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

### 2.4.1 Typesetting Markdown

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

```
1961 \let\startmarkdown\relax
1962 \let\stopmarkdown\relax
```

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

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

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

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

## 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 T<sub>E</sub>X *token renderers* is performed by the Lua layer. The plain T<sub>E</sub>X layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X 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 T<sub>E</sub>X, and `extension` 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 T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
1963 local upper, gsub, format, length =
1964   string.upper, string.gsub, string.format, string.len
1965 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
1966   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
1967   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.

```
1968 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.

```
1969 function util.err(msg, exit_code)
1970   io.stderr:write("markdown.lua: " .. msg .. "\n")
1971   os.exit(exit_code or 1)
1972 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.

```
1973 function util.cache(dir, string, salt, transform, suffix)
1974   local digest = md5.sumhexa(string .. (salt or ""))
1975   local name = util.pathname(dir, digest .. suffix)
1976   local file = io.open(name, "r")
1977   if file == nil then -- If no cache entry exists, then create a new one.
1978     file = assert(io.open(name, "w"),
1979       [[could not open file ]] .. name .. [[ for writing]])
1980     local result = string
1981     if transform ~= nil then
1982       result = transform(result)
1983     end
1984     assert(file:write(result))
```



```

1985     assert(file:close())
1986   end
1987   return name
1988 end

```

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

```

1989 function util.table_copy(t)
1990   local u = { }
1991   for k, v in pairs(t) do u[k] = v end
1992   return setmetatable(u, getmetatable(t))
1993 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [9, Chapter 21].

```

1994 function util.expand_tabs_in_line(s, tabstop)
1995   local tab = tabstop or 4
1996   local corr = 0
1997   return (s:gsub("\t", function(p)
1998     local sp = tab - (p - 1 + corr) % tab
1999     corr = corr - 1 + sp
2000     return string.rep(" ", sp)
2001   end))
2002 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.

```

2003 function util.walk(t, f)
2004   local typ = type(t)
2005   if typ == "string" then
2006     f(t)
2007   elseif typ == "table" then
2008     local i = 1
2009     local n
2010     n = t[i]
2011     while n do
2012       util.walk(n, f)
2013       i = i + 1
2014       n = t[i]
2015     end
2016   elseif typ == "function" then
2017     local ok, val = pcall(t)
2018     if ok then
2019       util.walk(val, f)
2020     end

```

```

2021 else
2022     f(tostring(t))
2023 end
2024 end

```

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

```

2025 function util.flatten(ary)
2026     local new = {}
2027     for _,v in ipairs(ary) do
2028         if type(v) == "table" then
2029             for _,w in ipairs(util.flatten(v)) do
2030                 new[#new + 1] = w
2031             end
2032         else
2033             new[#new + 1] = v
2034         end
2035     end
2036     return new
2037 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.

```

2038 function util.rope_to_string(rope)
2039     local buffer = {}
2040     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2041     return table.concat(buffer)
2042 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.

```

2043 function util.rope_last(rope)
2044     if #rope == 0 then
2045         return nil
2046     else
2047         local l = rope[#rope]
2048         if type(l) == "table" then
2049             return util.rope_last(l)
2050         else
2051             return l
2052         end
2053     end
2054 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  $1 \leq i \leq \#ary$ .

```

2055 function util.intersperse(ary, x)

```

```

2056 local new = {}
2057 local l = #ary
2058 for i,v in ipairs(ary) do
2059     local n = #new
2060     new[n + 1] = v
2061     if i ~= l then
2062         new[n + 2] = x
2063     end
2064 end
2065 return new
2066 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  $1 \leq i \leq \#ary$ .

```

2067 function util.map(ary, f)
2068     local new = {}
2069     for i,v in ipairs(ary) do
2070         new[i] = f(v)
2071     end
2072     return new
2073 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.

```

2074 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

2075     local char_escapes_list = ""
2076     for i,_ in pairs(char_escapes) do
2077         char_escapes_list = char_escapes_list .. i
2078     end

```

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

```

2079     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.

```

2080 if string_escapes then
2081     for k,v in pairs(string_escapes) do
2082         escapable = P(k) / v + escapable
2083     end
2084 end

```

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

```

2085 local escape_string = Cs((escapable + any)^0)

```

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

```

2086 return function(s)
2087     return lpeg.match(escape_string, s)
2088 end
2089 end

```

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

```

2090 function util.pathname(dir, file)
2091     if #dir == 0 then
2092         return file
2093     else
2094         return dir .. "/" .. file
2095     end
2096 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.

```

2097 local entities = {}
2098
2099 local character_entities = {
2100     ["Tab"] = 9,
2101     ["NewLine"] = 10,
2102     ["excl"] = 33,
2103     ["quot"] = 34,
2104     ["QUOT"] = 34,
2105     ["num"] = 35,
2106     ["dollar"] = 36,
2107     ["percent"] = 37,
2108     ["amp"] = 38,
2109     ["AMP"] = 38,
2110     ["apos"] = 39,
2111     ["lpar"] = 40,

```

2112 ["rpar"] = 41,  
2113 ["ast"] = 42,  
2114 ["midast"] = 42,  
2115 ["plus"] = 43,  
2116 ["comma"] = 44,  
2117 ["period"] = 46,  
2118 ["sol"] = 47,  
2119 ["colon"] = 58,  
2120 ["semi"] = 59,  
2121 ["lt"] = 60,  
2122 ["LT"] = 60,  
2123 ["equals"] = 61,  
2124 ["gt"] = 62,  
2125 ["GT"] = 62,  
2126 ["quest"] = 63,  
2127 ["commat"] = 64,  
2128 ["lsqb"] = 91,  
2129 ["lbrack"] = 91,  
2130 ["bsol"] = 92,  
2131 ["rsqb"] = 93,  
2132 ["rbrack"] = 93,  
2133 ["Hat"] = 94,  
2134 ["lowbar"] = 95,  
2135 ["grave"] = 96,  
2136 ["DiacriticalGrave"] = 96,  
2137 ["lcub"] = 123,  
2138 ["lbrace"] = 123,  
2139 ["verbar"] = 124,  
2140 ["vert"] = 124,  
2141 ["VerticalLine"] = 124,  
2142 ["rcub"] = 125,  
2143 ["rbrace"] = 125,  
2144 ["nbsp"] = 160,  
2145 ["NonBreakingSpace"] = 160,  
2146 ["iexcl"] = 161,  
2147 ["cent"] = 162,  
2148 ["pound"] = 163,  
2149 ["curren"] = 164,  
2150 ["yen"] = 165,  
2151 ["brvbar"] = 166,  
2152 ["sect"] = 167,  
2153 ["Dot"] = 168,  
2154 ["die"] = 168,  
2155 ["DoubleDot"] = 168,  
2156 ["uml"] = 168,  
2157 ["copy"] = 169,  
2158 ["COPY"] = 169,

2159 ["ordf"] = 170,  
 2160 ["laquo"] = 171,  
 2161 ["not"] = 172,  
 2162 ["shy"] = 173,  
 2163 ["reg"] = 174,  
 2164 ["circledR"] = 174,  
 2165 ["REG"] = 174,  
 2166 ["macr"] = 175,  
 2167 ["OverBar"] = 175,  
 2168 ["strns"] = 175,  
 2169 ["deg"] = 176,  
 2170 ["plusmn"] = 177,  
 2171 ["pm"] = 177,  
 2172 ["PlusMinus"] = 177,  
 2173 ["sup2"] = 178,  
 2174 ["sup3"] = 179,  
 2175 ["acute"] = 180,  
 2176 ["DiacriticalAcute"] = 180,  
 2177 ["micro"] = 181,  
 2178 ["para"] = 182,  
 2179 ["middot"] = 183,  
 2180 ["centerdot"] = 183,  
 2181 ["CenterDot"] = 183,  
 2182 ["cedil"] = 184,  
 2183 ["Cedilla"] = 184,  
 2184 ["sup1"] = 185,  
 2185 ["ordm"] = 186,  
 2186 ["raquo"] = 187,  
 2187 ["frac14"] = 188,  
 2188 ["frac12"] = 189,  
 2189 ["half"] = 189,  
 2190 ["frac34"] = 190,  
 2191 ["iquest"] = 191,  
 2192 ["Agrave"] = 192,  
 2193 ["Aacute"] = 193,  
 2194 ["Acirc"] = 194,  
 2195 ["Atilde"] = 195,  
 2196 ["Auml"] = 196,  
 2197 ["Aring"] = 197,  
 2198 ["AElig"] = 198,  
 2199 ["Ccedil"] = 199,  
 2200 ["Egrave"] = 200,  
 2201 ["Eacute"] = 201,  
 2202 ["Ecirc"] = 202,  
 2203 ["Euml"] = 203,  
 2204 ["Igrave"] = 204,  
 2205 ["Iacute"] = 205,

2206 ["Icirc"] = 206,  
2207 ["Iuml"] = 207,  
2208 ["ETH"] = 208,  
2209 ["Ntilde"] = 209,  
2210 ["Ograve"] = 210,  
2211 ["Oacute"] = 211,  
2212 ["Ocirc"] = 212,  
2213 ["Otilde"] = 213,  
2214 ["Ouml"] = 214,  
2215 ["times"] = 215,  
2216 ["Oslash"] = 216,  
2217 ["Ugrave"] = 217,  
2218 ["Uacute"] = 218,  
2219 ["Ucirc"] = 219,  
2220 ["Uuml"] = 220,  
2221 ["Yacute"] = 221,  
2222 ["THORN"] = 222,  
2223 ["szlig"] = 223,  
2224 ["agrave"] = 224,  
2225 ["aacute"] = 225,  
2226 ["acirc"] = 226,  
2227 ["atilde"] = 227,  
2228 ["auml"] = 228,  
2229 ["aring"] = 229,  
2230 ["aelig"] = 230,  
2231 ["ccedil"] = 231,  
2232 ["egrave"] = 232,  
2233 ["eacute"] = 233,  
2234 ["ecirc"] = 234,  
2235 ["euml"] = 235,  
2236 ["igrave"] = 236,  
2237 ["iacute"] = 237,  
2238 ["icirc"] = 238,  
2239 ["iuml"] = 239,  
2240 ["eth"] = 240,  
2241 ["ntilde"] = 241,  
2242 ["ograve"] = 242,  
2243 ["oacute"] = 243,  
2244 ["ocirc"] = 244,  
2245 ["otilde"] = 245,  
2246 ["ouml"] = 246,  
2247 ["divide"] = 247,  
2248 ["div"] = 247,  
2249 ["oslash"] = 248,  
2250 ["ugrave"] = 249,  
2251 ["uacute"] = 250,  
2252 ["ucirc"] = 251,

2253 ["uuml"] = 252,  
2254 ["yacute"] = 253,  
2255 ["thorn"] = 254,  
2256 ["yuml"] = 255,  
2257 ["Amacr"] = 256,  
2258 ["amacr"] = 257,  
2259 ["Abreve"] = 258,  
2260 ["abreve"] = 259,  
2261 ["Aogon"] = 260,  
2262 ["aogon"] = 261,  
2263 ["Cacute"] = 262,  
2264 ["cacute"] = 263,  
2265 ["Ccirc"] = 264,  
2266 ["ccirc"] = 265,  
2267 ["Cdot"] = 266,  
2268 ["cdot"] = 267,  
2269 ["Ccaron"] = 268,  
2270 ["ccaron"] = 269,  
2271 ["Dcaron"] = 270,  
2272 ["dcaron"] = 271,  
2273 ["Dstrok"] = 272,  
2274 ["dstrok"] = 273,  
2275 ["Emacr"] = 274,  
2276 ["emacr"] = 275,  
2277 ["Edot"] = 278,  
2278 ["edot"] = 279,  
2279 ["Eogon"] = 280,  
2280 ["eogon"] = 281,  
2281 ["Ecaron"] = 282,  
2282 ["ecaron"] = 283,  
2283 ["Gcirc"] = 284,  
2284 ["gcirc"] = 285,  
2285 ["Gbreve"] = 286,  
2286 ["gbreve"] = 287,  
2287 ["Gdot"] = 288,  
2288 ["gdot"] = 289,  
2289 ["Gcedil"] = 290,  
2290 ["Hcirc"] = 292,  
2291 ["hcirc"] = 293,  
2292 ["Hstrok"] = 294,  
2293 ["hstrok"] = 295,  
2294 ["Itilde"] = 296,  
2295 ["itilde"] = 297,  
2296 ["Imacr"] = 298,  
2297 ["imacr"] = 299,  
2298 ["Iogon"] = 302,  
2299 ["iogon"] = 303,



2300 ["Idot"] = 304,  
2301 ["imath"] = 305,  
2302 ["inodot"] = 305,  
2303 ["IJlig"] = 306,  
2304 ["ijlig"] = 307,  
2305 ["Jcirc"] = 308,  
2306 ["jcirc"] = 309,  
2307 ["Kcedil"] = 310,  
2308 ["kcedil"] = 311,  
2309 ["kgreen"] = 312,  
2310 ["Lacute"] = 313,  
2311 ["lacute"] = 314,  
2312 ["Lcedil"] = 315,  
2313 ["lcedil"] = 316,  
2314 ["Lcaron"] = 317,  
2315 ["lcaron"] = 318,  
2316 ["Lmidot"] = 319,  
2317 ["lmidot"] = 320,  
2318 ["Lstrok"] = 321,  
2319 ["lstrok"] = 322,  
2320 ["Nacute"] = 323,  
2321 ["nacute"] = 324,  
2322 ["Ncedil"] = 325,  
2323 ["ncedil"] = 326,  
2324 ["Ncaron"] = 327,  
2325 ["ncaron"] = 328,  
2326 ["napos"] = 329,  
2327 ["ENG"] = 330,  
2328 ["eng"] = 331,  
2329 ["Omacr"] = 332,  
2330 ["omacr"] = 333,  
2331 ["Odblac"] = 336,  
2332 ["odblac"] = 337,  
2333 ["OElig"] = 338,  
2334 ["oelig"] = 339,  
2335 ["Racute"] = 340,  
2336 ["racute"] = 341,  
2337 ["Rcedil"] = 342,  
2338 ["rcedil"] = 343,  
2339 ["Rcaron"] = 344,  
2340 ["rcaron"] = 345,  
2341 ["Sacute"] = 346,  
2342 ["sacute"] = 347,  
2343 ["Scirc"] = 348,  
2344 ["scirc"] = 349,  
2345 ["Scedil"] = 350,  
2346 ["scedil"] = 351,

2347 ["Scaron"] = 352,  
2348 ["scaron"] = 353,  
2349 ["Tcedil"] = 354,  
2350 ["tcedil"] = 355,  
2351 ["Tcaron"] = 356,  
2352 ["tcaron"] = 357,  
2353 ["Tstrok"] = 358,  
2354 ["tstrok"] = 359,  
2355 ["Utilde"] = 360,  
2356 ["utilde"] = 361,  
2357 ["Umacr"] = 362,  
2358 ["umacr"] = 363,  
2359 ["Ubreve"] = 364,  
2360 ["ubreve"] = 365,  
2361 ["Uring"] = 366,  
2362 ["uring"] = 367,  
2363 ["Udblac"] = 368,  
2364 ["udblac"] = 369,  
2365 ["Uogon"] = 370,  
2366 ["uogon"] = 371,  
2367 ["Wcirc"] = 372,  
2368 ["wcirc"] = 373,  
2369 ["Ycirc"] = 374,  
2370 ["ycirc"] = 375,  
2371 ["Yuml"] = 376,  
2372 ["Zacute"] = 377,  
2373 ["zacute"] = 378,  
2374 ["Zdot"] = 379,  
2375 ["zdot"] = 380,  
2376 ["Zcaron"] = 381,  
2377 ["zcaron"] = 382,  
2378 ["fnof"] = 402,  
2379 ["imped"] = 437,  
2380 ["gacute"] = 501,  
2381 ["jmath"] = 567,  
2382 ["circ"] = 710,  
2383 ["caron"] = 711,  
2384 ["Hacek"] = 711,  
2385 ["breve"] = 728,  
2386 ["Breve"] = 728,  
2387 ["dot"] = 729,  
2388 ["DiacriticalDot"] = 729,  
2389 ["ring"] = 730,  
2390 ["ogon"] = 731,  
2391 ["tilde"] = 732,  
2392 ["DiacriticalTilde"] = 732,  
2393 ["dblac"] = 733,

2394 ["DiacriticalDoubleAcute"] = 733,  
2395 ["DownBreve"] = 785,  
2396 ["UnderBar"] = 818,  
2397 ["Alpha"] = 913,  
2398 ["Beta"] = 914,  
2399 ["Gamma"] = 915,  
2400 ["Delta"] = 916,  
2401 ["Epsilon"] = 917,  
2402 ["Zeta"] = 918,  
2403 ["Eta"] = 919,  
2404 ["Theta"] = 920,  
2405 ["Iota"] = 921,  
2406 ["Kappa"] = 922,  
2407 ["Lambda"] = 923,  
2408 ["Mu"] = 924,  
2409 ["Nu"] = 925,  
2410 ["Xi"] = 926,  
2411 ["Omicron"] = 927,  
2412 ["Pi"] = 928,  
2413 ["Rho"] = 929,  
2414 ["Sigma"] = 931,  
2415 ["Tau"] = 932,  
2416 ["Upsilon"] = 933,  
2417 ["Phi"] = 934,  
2418 ["Chi"] = 935,  
2419 ["Psi"] = 936,  
2420 ["Omega"] = 937,  
2421 ["alpha"] = 945,  
2422 ["beta"] = 946,  
2423 ["gamma"] = 947,  
2424 ["delta"] = 948,  
2425 ["epsiv"] = 949,  
2426 ["varepsilon"] = 949,  
2427 ["epsilon"] = 949,  
2428 ["zeta"] = 950,  
2429 ["eta"] = 951,  
2430 ["theta"] = 952,  
2431 ["iota"] = 953,  
2432 ["kappa"] = 954,  
2433 ["lambda"] = 955,  
2434 ["mu"] = 956,  
2435 ["nu"] = 957,  
2436 ["xi"] = 958,  
2437 ["omicron"] = 959,  
2438 ["pi"] = 960,  
2439 ["rho"] = 961,  
2440 ["sigmav"] = 962,

2441 ["varsigma"] = 962,  
2442 ["sigmaf"] = 962,  
2443 ["sigma"] = 963,  
2444 ["tau"] = 964,  
2445 ["upsilon"] = 965,  
2446 ["upsilon"] = 965,  
2447 ["phi"] = 966,  
2448 ["phiv"] = 966,  
2449 ["varphi"] = 966,  
2450 ["chi"] = 967,  
2451 ["psi"] = 968,  
2452 ["omega"] = 969,  
2453 ["thetav"] = 977,  
2454 ["vartheta"] = 977,  
2455 ["thetasym"] = 977,  
2456 ["Upsilon"] = 978,  
2457 ["upsih"] = 978,  
2458 ["straightphi"] = 981,  
2459 ["piv"] = 982,  
2460 ["varpi"] = 982,  
2461 ["Gammad"] = 988,  
2462 ["gammad"] = 989,  
2463 ["digamma"] = 989,  
2464 ["kappav"] = 1008,  
2465 ["varkappa"] = 1008,  
2466 ["rhov"] = 1009,  
2467 ["varrho"] = 1009,  
2468 ["epsi"] = 1013,  
2469 ["straightepsilon"] = 1013,  
2470 ["bepsi"] = 1014,  
2471 ["backepsilon"] = 1014,  
2472 ["IOcy"] = 1025,  
2473 ["DJcy"] = 1026,  
2474 ["GJcy"] = 1027,  
2475 ["Jukcy"] = 1028,  
2476 ["DScy"] = 1029,  
2477 ["Iukcy"] = 1030,  
2478 ["YIcy"] = 1031,  
2479 ["Jsercy"] = 1032,  
2480 ["LJcy"] = 1033,  
2481 ["NJcy"] = 1034,  
2482 ["TSHcy"] = 1035,  
2483 ["KJcy"] = 1036,  
2484 ["Ubrcy"] = 1038,  
2485 ["DZcy"] = 1039,  
2486 ["Acy"] = 1040,  
2487 ["Bcy"] = 1041,

2488 ["Vcy"] = 1042,  
2489 ["Gcy"] = 1043,  
2490 ["Dcy"] = 1044,  
2491 ["IEcy"] = 1045,  
2492 ["ZHcy"] = 1046,  
2493 ["Zcy"] = 1047,  
2494 ["Icy"] = 1048,  
2495 ["Jcy"] = 1049,  
2496 ["Kcy"] = 1050,  
2497 ["Lcy"] = 1051,  
2498 ["Mcy"] = 1052,  
2499 ["Ncy"] = 1053,  
2500 ["Ocy"] = 1054,  
2501 ["Pcy"] = 1055,  
2502 ["Rcy"] = 1056,  
2503 ["Scy"] = 1057,  
2504 ["Tcy"] = 1058,  
2505 ["Ucy"] = 1059,  
2506 ["Fcy"] = 1060,  
2507 ["KHcy"] = 1061,  
2508 ["TScy"] = 1062,  
2509 ["CHcy"] = 1063,  
2510 ["SHcy"] = 1064,  
2511 ["SHCHcy"] = 1065,  
2512 ["HARDcy"] = 1066,  
2513 ["Ycy"] = 1067,  
2514 ["SOFTcy"] = 1068,  
2515 ["Ecy"] = 1069,  
2516 ["YUcy"] = 1070,  
2517 ["YAcy"] = 1071,  
2518 ["acy"] = 1072,  
2519 ["bcy"] = 1073,  
2520 ["vcy"] = 1074,  
2521 ["gcy"] = 1075,  
2522 ["dcy"] = 1076,  
2523 ["iecy"] = 1077,  
2524 ["zhcy"] = 1078,  
2525 ["zcy"] = 1079,  
2526 ["icy"] = 1080,  
2527 ["jcy"] = 1081,  
2528 ["kcy"] = 1082,  
2529 ["lcy"] = 1083,  
2530 ["mcy"] = 1084,  
2531 ["ncy"] = 1085,  
2532 ["ocy"] = 1086,  
2533 ["pcy"] = 1087,  
2534 ["rcy"] = 1088,

2535 ["scy"] = 1089,  
2536 ["tcy"] = 1090,  
2537 ["ucy"] = 1091,  
2538 ["fcy"] = 1092,  
2539 ["khcy"] = 1093,  
2540 ["tscy"] = 1094,  
2541 ["chcy"] = 1095,  
2542 ["shcy"] = 1096,  
2543 ["shchcy"] = 1097,  
2544 ["hardcy"] = 1098,  
2545 ["ycy"] = 1099,  
2546 ["softcy"] = 1100,  
2547 ["ecy"] = 1101,  
2548 ["yucy"] = 1102,  
2549 ["yacy"] = 1103,  
2550 ["iocy"] = 1105,  
2551 ["djcy"] = 1106,  
2552 ["gjcy"] = 1107,  
2553 ["jukcy"] = 1108,  
2554 ["dscy"] = 1109,  
2555 ["iukcy"] = 1110,  
2556 ["yicy"] = 1111,  
2557 ["jsercy"] = 1112,  
2558 ["ljcy"] = 1113,  
2559 ["njcy"] = 1114,  
2560 ["tshcy"] = 1115,  
2561 ["kjcy"] = 1116,  
2562 ["ubrscy"] = 1118,  
2563 ["dzcy"] = 1119,  
2564 ["ensp"] = 8194,  
2565 ["emsp"] = 8195,  
2566 ["emsp13"] = 8196,  
2567 ["emsp14"] = 8197,  
2568 ["numsp"] = 8199,  
2569 ["puncsp"] = 8200,  
2570 ["thinsp"] = 8201,  
2571 ["ThinSpace"] = 8201,  
2572 ["hairsp"] = 8202,  
2573 ["VeryThinSpace"] = 8202,  
2574 ["ZeroWidthSpace"] = 8203,  
2575 ["NegativeVeryThinSpace"] = 8203,  
2576 ["NegativeThinSpace"] = 8203,  
2577 ["NegativeMediumSpace"] = 8203,  
2578 ["NegativeThickSpace"] = 8203,  
2579 ["zwnj"] = 8204,  
2580 ["zwj"] = 8205,  
2581 ["lrm"] = 8206,

2582 ["rlm"] = 8207,  
2583 ["hyphen"] = 8208,  
2584 ["dash"] = 8208,  
2585 ["ndash"] = 8211,  
2586 ["mdash"] = 8212,  
2587 ["horbar"] = 8213,  
2588 ["Verbar"] = 8214,  
2589 ["Vert"] = 8214,  
2590 ["lsquo"] = 8216,  
2591 ["OpenCurlyQuote"] = 8216,  
2592 ["rsquo"] = 8217,  
2593 ["rsquor"] = 8217,  
2594 ["CloseCurlyQuote"] = 8217,  
2595 ["lsquor"] = 8218,  
2596 ["sbquo"] = 8218,  
2597 ["ldquo"] = 8220,  
2598 ["OpenCurlyDoubleQuote"] = 8220,  
2599 ["rdquo"] = 8221,  
2600 ["rdquor"] = 8221,  
2601 ["CloseCurlyDoubleQuote"] = 8221,  
2602 ["ldquor"] = 8222,  
2603 ["bdquo"] = 8222,  
2604 ["dagger"] = 8224,  
2605 ["Dagger"] = 8225,  
2606 ["ddagger"] = 8225,  
2607 ["bull"] = 8226,  
2608 ["bullet"] = 8226,  
2609 ["nldr"] = 8229,  
2610 ["hellip"] = 8230,  
2611 ["mldr"] = 8230,  
2612 ["permil"] = 8240,  
2613 ["pertenk"] = 8241,  
2614 ["prime"] = 8242,  
2615 ["Prime"] = 8243,  
2616 ["tprime"] = 8244,  
2617 ["bprime"] = 8245,  
2618 ["backprime"] = 8245,  
2619 ["lsaquo"] = 8249,  
2620 ["rsaquo"] = 8250,  
2621 ["oline"] = 8254,  
2622 ["caret"] = 8257,  
2623 ["hybull"] = 8259,  
2624 ["frasl"] = 8260,  
2625 ["bsemi"] = 8271,  
2626 ["qprime"] = 8279,  
2627 ["MediumSpace"] = 8287,  
2628 ["NoBreak"] = 8288,

2629 ["ApplyFunction"] = 8289,  
2630 ["af"] = 8289,  
2631 ["InvisibleTimes"] = 8290,  
2632 ["it"] = 8290,  
2633 ["InvisibleComma"] = 8291,  
2634 ["ic"] = 8291,  
2635 ["euro"] = 8364,  
2636 ["tdot"] = 8411,  
2637 ["TripleDot"] = 8411,  
2638 ["DotDot"] = 8412,  
2639 ["Copf"] = 8450,  
2640 ["complexes"] = 8450,  
2641 ["incare"] = 8453,  
2642 ["gscr"] = 8458,  
2643 ["hamilt"] = 8459,  
2644 ["HilbertSpace"] = 8459,  
2645 ["Hscr"] = 8459,  
2646 ["Hfr"] = 8460,  
2647 ["Poincareplane"] = 8460,  
2648 ["quaternions"] = 8461,  
2649 ["Hopf"] = 8461,  
2650 ["planckh"] = 8462,  
2651 ["planck"] = 8463,  
2652 ["hbar"] = 8463,  
2653 ["plankv"] = 8463,  
2654 ["hslash"] = 8463,  
2655 ["Iscr"] = 8464,  
2656 ["imagline"] = 8464,  
2657 ["image"] = 8465,  
2658 ["Im"] = 8465,  
2659 ["imagpart"] = 8465,  
2660 ["Ifr"] = 8465,  
2661 ["Lscr"] = 8466,  
2662 ["lagran"] = 8466,  
2663 ["Laplacetrif"] = 8466,  
2664 ["ell"] = 8467,  
2665 ["Nopf"] = 8469,  
2666 ["naturals"] = 8469,  
2667 ["numero"] = 8470,  
2668 ["copysr"] = 8471,  
2669 ["weierp"] = 8472,  
2670 ["wp"] = 8472,  
2671 ["Popf"] = 8473,  
2672 ["primes"] = 8473,  
2673 ["rationals"] = 8474,  
2674 ["Qopf"] = 8474,  
2675 ["Rscr"] = 8475,



2676 ["realine"] = 8475,  
2677 ["real"] = 8476,  
2678 ["Re"] = 8476,  
2679 ["realpart"] = 8476,  
2680 ["Rfr"] = 8476,  
2681 ["reals"] = 8477,  
2682 ["Ropf"] = 8477,  
2683 ["rx"] = 8478,  
2684 ["trade"] = 8482,  
2685 ["TRADE"] = 8482,  
2686 ["integers"] = 8484,  
2687 ["Zopf"] = 8484,  
2688 ["ohm"] = 8486,  
2689 ["mho"] = 8487,  
2690 ["Zfr"] = 8488,  
2691 ["zeetrf"] = 8488,  
2692 ["iiota"] = 8489,  
2693 ["angst"] = 8491,  
2694 ["bernou"] = 8492,  
2695 ["Bernoullis"] = 8492,  
2696 ["Bscr"] = 8492,  
2697 ["Cfr"] = 8493,  
2698 ["Cayleys"] = 8493,  
2699 ["escr"] = 8495,  
2700 ["Escr"] = 8496,  
2701 ["expectation"] = 8496,  
2702 ["Fscr"] = 8497,  
2703 ["Fouriertrf"] = 8497,  
2704 ["phmmat"] = 8499,  
2705 ["Mellintrf"] = 8499,  
2706 ["Mscr"] = 8499,  
2707 ["order"] = 8500,  
2708 ["orderof"] = 8500,  
2709 ["oscr"] = 8500,  
2710 ["alefsym"] = 8501,  
2711 ["aleph"] = 8501,  
2712 ["beth"] = 8502,  
2713 ["gimel"] = 8503,  
2714 ["daleth"] = 8504,  
2715 ["CapitalDifferentialD"] = 8517,  
2716 ["DD"] = 8517,  
2717 ["DifferentialD"] = 8518,  
2718 ["dd"] = 8518,  
2719 ["ExponentialE"] = 8519,  
2720 ["exponentiale"] = 8519,  
2721 ["ee"] = 8519,  
2722 ["ImaginaryI"] = 8520,

2723 ["ii"] = 8520,  
 2724 ["frac13"] = 8531,  
 2725 ["frac23"] = 8532,  
 2726 ["frac15"] = 8533,  
 2727 ["frac25"] = 8534,  
 2728 ["frac35"] = 8535,  
 2729 ["frac45"] = 8536,  
 2730 ["frac16"] = 8537,  
 2731 ["frac56"] = 8538,  
 2732 ["frac18"] = 8539,  
 2733 ["frac38"] = 8540,  
 2734 ["frac58"] = 8541,  
 2735 ["frac78"] = 8542,  
 2736 ["larr"] = 8592,  
 2737 ["leftarrow"] = 8592,  
 2738 ["LeftArrow"] = 8592,  
 2739 ["slarr"] = 8592,  
 2740 ["ShortLeftArrow"] = 8592,  
 2741 ["uarr"] = 8593,  
 2742 ["uparrow"] = 8593,  
 2743 ["UpArrow"] = 8593,  
 2744 ["ShortUpArrow"] = 8593,  
 2745 ["rarr"] = 8594,  
 2746 ["rightarrow"] = 8594,  
 2747 ["RightArrow"] = 8594,  
 2748 ["srarr"] = 8594,  
 2749 ["ShortRightArrow"] = 8594,  
 2750 ["darr"] = 8595,  
 2751 ["downarrow"] = 8595,  
 2752 ["DownArrow"] = 8595,  
 2753 ["ShortDownArrow"] = 8595,  
 2754 ["harr"] = 8596,  
 2755 ["leftrightarrow"] = 8596,  
 2756 ["LeftRightArrow"] = 8596,  
 2757 ["varr"] = 8597,  
 2758 ["updownarrow"] = 8597,  
 2759 ["UpDownArrow"] = 8597,  
 2760 ["nwarr"] = 8598,  
 2761 ["UpperLeftArrow"] = 8598,  
 2762 ["nwarrow"] = 8598,  
 2763 ["nearr"] = 8599,  
 2764 ["UpperRightArrow"] = 8599,  
 2765 ["nearrow"] = 8599,  
 2766 ["searr"] = 8600,  
 2767 ["searrow"] = 8600,  
 2768 ["LowerRightArrow"] = 8600,  
 2769 ["swarr"] = 8601,

2770 ["swarrow"] = 8601,  
 2771 ["LowerLeftArrow"] = 8601,  
 2772 ["nlarr"] = 8602,  
 2773 ["nleftarrow"] = 8602,  
 2774 ["nrarr"] = 8603,  
 2775 ["nrightarrow"] = 8603,  
 2776 ["rarrw"] = 8605,  
 2777 ["rightsquigarrow"] = 8605,  
 2778 ["Larr"] = 8606,  
 2779 ["twoheadleftarrow"] = 8606,  
 2780 ["Uarr"] = 8607,  
 2781 ["Rarr"] = 8608,  
 2782 ["twoheadrightarrow"] = 8608,  
 2783 ["Darr"] = 8609,  
 2784 ["larrtl"] = 8610,  
 2785 ["leftarrowtail"] = 8610,  
 2786 ["rarrtl"] = 8611,  
 2787 ["rightarrowtail"] = 8611,  
 2788 ["LeftTeeArrow"] = 8612,  
 2789 ["mapstoleft"] = 8612,  
 2790 ["UpTeeArrow"] = 8613,  
 2791 ["mapstoup"] = 8613,  
 2792 ["map"] = 8614,  
 2793 ["RightTeeArrow"] = 8614,  
 2794 ["mapsto"] = 8614,  
 2795 ["DownTeeArrow"] = 8615,  
 2796 ["mapstodown"] = 8615,  
 2797 ["larrhk"] = 8617,  
 2798 ["hookleftarrow"] = 8617,  
 2799 ["rarrhk"] = 8618,  
 2800 ["hookrightarrow"] = 8618,  
 2801 ["larrlp"] = 8619,  
 2802 ["looparrowleft"] = 8619,  
 2803 ["rarrlp"] = 8620,  
 2804 ["looparrowright"] = 8620,  
 2805 ["harrw"] = 8621,  
 2806 ["leftrightsquigarrow"] = 8621,  
 2807 ["nharr"] = 8622,  
 2808 ["nleftrightarrow"] = 8622,  
 2809 ["lsh"] = 8624,  
 2810 ["Lsh"] = 8624,  
 2811 ["rsh"] = 8625,  
 2812 ["Rsh"] = 8625,  
 2813 ["ldsh"] = 8626,  
 2814 ["rdsh"] = 8627,  
 2815 ["crarr"] = 8629,  
 2816 ["cularr"] = 8630,

2817 ["curvearrowleft"] = 8630,  
 2818 ["curarr"] = 8631,  
 2819 ["curvearrowright"] = 8631,  
 2820 ["olarr"] = 8634,  
 2821 ["circlearrowleft"] = 8634,  
 2822 ["orarr"] = 8635,  
 2823 ["circlearrowright"] = 8635,  
 2824 ["lharu"] = 8636,  
 2825 ["LeftVector"] = 8636,  
 2826 ["leftharpoonup"] = 8636,  
 2827 ["lhard"] = 8637,  
 2828 ["leftharpoondown"] = 8637,  
 2829 ["DownLeftVector"] = 8637,  
 2830 ["uharr"] = 8638,  
 2831 ["upharpoonright"] = 8638,  
 2832 ["RightUpVector"] = 8638,  
 2833 ["uharl"] = 8639,  
 2834 ["upharpoonleft"] = 8639,  
 2835 ["LeftUpVector"] = 8639,  
 2836 ["rharu"] = 8640,  
 2837 ["RightVector"] = 8640,  
 2838 ["rightharpoonup"] = 8640,  
 2839 ["rhard"] = 8641,  
 2840 ["rightharpoondown"] = 8641,  
 2841 ["DownRightVector"] = 8641,  
 2842 ["dharr"] = 8642,  
 2843 ["RightDownVector"] = 8642,  
 2844 ["downharpoonright"] = 8642,  
 2845 ["dharl"] = 8643,  
 2846 ["LeftDownVector"] = 8643,  
 2847 ["downharpoonleft"] = 8643,  
 2848 ["rlarr"] = 8644,  
 2849 ["rightleftarrows"] = 8644,  
 2850 ["RightArrowLeftArrow"] = 8644,  
 2851 ["udarr"] = 8645,  
 2852 ["UpArrowDownArrow"] = 8645,  
 2853 ["lrarr"] = 8646,  
 2854 ["leftrightarrows"] = 8646,  
 2855 ["LeftArrowRightArrow"] = 8646,  
 2856 ["llarr"] = 8647,  
 2857 ["leftleftarrows"] = 8647,  
 2858 ["uuarr"] = 8648,  
 2859 ["upuparrows"] = 8648,  
 2860 ["rrarr"] = 8649,  
 2861 ["rightrightarrows"] = 8649,  
 2862 ["ddarr"] = 8650,  
 2863 ["downdownarrows"] = 8650,

2864 ["lrhar"] = 8651,  
 2865 ["ReverseEquilibrium"] = 8651,  
 2866 ["leftrightharpoons"] = 8651,  
 2867 ["rlhar"] = 8652,  
 2868 ["rightleftharpoons"] = 8652,  
 2869 ["Equilibrium"] = 8652,  
 2870 ["nLArr"] = 8653,  
 2871 ["nLeftarrow"] = 8653,  
 2872 ["nhArr"] = 8654,  
 2873 ["nLeftrightarrow"] = 8654,  
 2874 ["nrArr"] = 8655,  
 2875 ["nRrightarrow"] = 8655,  
 2876 ["lArr"] = 8656,  
 2877 ["Leftarrow"] = 8656,  
 2878 ["DoubleLeftArrow"] = 8656,  
 2879 ["uArr"] = 8657,  
 2880 ["Uparrow"] = 8657,  
 2881 ["DoubleUpArrow"] = 8657,  
 2882 ["rArr"] = 8658,  
 2883 ["Rrightarrow"] = 8658,  
 2884 ["Implies"] = 8658,  
 2885 ["DoubleRightArrow"] = 8658,  
 2886 ["dArr"] = 8659,  
 2887 ["Downarrow"] = 8659,  
 2888 ["DoubleDownArrow"] = 8659,  
 2889 ["hArr"] = 8660,  
 2890 ["Leftrightarrow"] = 8660,  
 2891 ["DoubleLeftRightArrow"] = 8660,  
 2892 ["iff"] = 8660,  
 2893 ["vArr"] = 8661,  
 2894 ["Updownarrow"] = 8661,  
 2895 ["DoubleUpDownArrow"] = 8661,  
 2896 ["nwArr"] = 8662,  
 2897 ["neArr"] = 8663,  
 2898 ["seArr"] = 8664,  
 2899 ["swArr"] = 8665,  
 2900 ["lAarr"] = 8666,  
 2901 ["Lleftarrow"] = 8666,  
 2902 ["rAarr"] = 8667,  
 2903 ["Rrightarrow"] = 8667,  
 2904 ["zigrarr"] = 8669,  
 2905 ["larrb"] = 8676,  
 2906 ["LeftArrowBar"] = 8676,  
 2907 ["rarrb"] = 8677,  
 2908 ["RightArrowBar"] = 8677,  
 2909 ["duarr"] = 8693,  
 2910 ["DownArrowUpArrow"] = 8693,

2911 ["loarr"] = 8701,  
 2912 ["roarr"] = 8702,  
 2913 ["hoarr"] = 8703,  
 2914 ["forall"] = 8704,  
 2915 ["ForAll"] = 8704,  
 2916 ["comp"] = 8705,  
 2917 ["complement"] = 8705,  
 2918 ["part"] = 8706,  
 2919 ["PartialD"] = 8706,  
 2920 ["exist"] = 8707,  
 2921 ["Exists"] = 8707,  
 2922 ["nexist"] = 8708,  
 2923 ["NotExists"] = 8708,  
 2924 ["nexists"] = 8708,  
 2925 ["empty"] = 8709,  
 2926 ["emptyset"] = 8709,  
 2927 ["emptyv"] = 8709,  
 2928 ["varnothing"] = 8709,  
 2929 ["nabla"] = 8711,  
 2930 ["Del"] = 8711,  
 2931 ["isin"] = 8712,  
 2932 ["isinv"] = 8712,  
 2933 ["Element"] = 8712,  
 2934 ["in"] = 8712,  
 2935 ["notin"] = 8713,  
 2936 ["NotElement"] = 8713,  
 2937 ["notinva"] = 8713,  
 2938 ["niv"] = 8715,  
 2939 ["ReverseElement"] = 8715,  
 2940 ["ni"] = 8715,  
 2941 ["SuchThat"] = 8715,  
 2942 ["notni"] = 8716,  
 2943 ["notniva"] = 8716,  
 2944 ["NotReverseElement"] = 8716,  
 2945 ["prod"] = 8719,  
 2946 ["Product"] = 8719,  
 2947 ["coprod"] = 8720,  
 2948 ["Coproduct"] = 8720,  
 2949 ["sum"] = 8721,  
 2950 ["Sum"] = 8721,  
 2951 ["minus"] = 8722,  
 2952 ["mnplus"] = 8723,  
 2953 ["mp"] = 8723,  
 2954 ["MinusPlus"] = 8723,  
 2955 ["plusdo"] = 8724,  
 2956 ["dotplus"] = 8724,  
 2957 ["setmn"] = 8726,

2958 ["setminus"] = 8726,  
 2959 ["Backslash"] = 8726,  
 2960 ["ssetmn"] = 8726,  
 2961 ["smallsetminus"] = 8726,  
 2962 ["lowast"] = 8727,  
 2963 ["compfn"] = 8728,  
 2964 ["SmallCircle"] = 8728,  
 2965 ["radic"] = 8730,  
 2966 ["Sqrt"] = 8730,  
 2967 ["prop"] = 8733,  
 2968 ["propto"] = 8733,  
 2969 ["Proportional"] = 8733,  
 2970 ["vprop"] = 8733,  
 2971 ["varpropto"] = 8733,  
 2972 ["infin"] = 8734,  
 2973 ["angrt"] = 8735,  
 2974 ["ang"] = 8736,  
 2975 ["angle"] = 8736,  
 2976 ["angmsd"] = 8737,  
 2977 ["measuredangle"] = 8737,  
 2978 ["angsph"] = 8738,  
 2979 ["mid"] = 8739,  
 2980 ["VerticalBar"] = 8739,  
 2981 ["smid"] = 8739,  
 2982 ["shortmid"] = 8739,  
 2983 ["nmid"] = 8740,  
 2984 ["NotVerticalBar"] = 8740,  
 2985 ["nsmid"] = 8740,  
 2986 ["nshortmid"] = 8740,  
 2987 ["par"] = 8741,  
 2988 ["parallel"] = 8741,  
 2989 ["DoubleVerticalBar"] = 8741,  
 2990 ["spar"] = 8741,  
 2991 ["shortparallel"] = 8741,  
 2992 ["npar"] = 8742,  
 2993 ["nparallel"] = 8742,  
 2994 ["NotDoubleVerticalBar"] = 8742,  
 2995 ["nspar"] = 8742,  
 2996 ["nshortparallel"] = 8742,  
 2997 ["and"] = 8743,  
 2998 ["wedge"] = 8743,  
 2999 ["or"] = 8744,  
 3000 ["vee"] = 8744,  
 3001 ["cap"] = 8745,  
 3002 ["cup"] = 8746,  
 3003 ["int"] = 8747,  
 3004 ["Integral"] = 8747,

3005 ["Int"] = 8748,  
3006 ["tint"] = 8749,  
3007 ["iiint"] = 8749,  
3008 ["conint"] = 8750,  
3009 ["oint"] = 8750,  
3010 ["ContourIntegral"] = 8750,  
3011 ["Conint"] = 8751,  
3012 ["DoubleContourIntegral"] = 8751,  
3013 ["Cconint"] = 8752,  
3014 ["cwint"] = 8753,  
3015 ["cwconint"] = 8754,  
3016 ["ClockwiseContourIntegral"] = 8754,  
3017 ["awconint"] = 8755,  
3018 ["CounterClockwiseContourIntegral"] = 8755,  
3019 ["there4"] = 8756,  
3020 ["therefore"] = 8756,  
3021 ["Therefore"] = 8756,  
3022 ["becaus"] = 8757,  
3023 ["because"] = 8757,  
3024 ["Because"] = 8757,  
3025 ["ratio"] = 8758,  
3026 ["Colon"] = 8759,  
3027 ["Proportion"] = 8759,  
3028 ["minusd"] = 8760,  
3029 ["dotminus"] = 8760,  
3030 ["mDDot"] = 8762,  
3031 ["homtht"] = 8763,  
3032 ["sim"] = 8764,  
3033 ["Tilde"] = 8764,  
3034 ["thksim"] = 8764,  
3035 ["thicksim"] = 8764,  
3036 ["bsim"] = 8765,  
3037 ["backsim"] = 8765,  
3038 ["ac"] = 8766,  
3039 ["mstpos"] = 8766,  
3040 ["acd"] = 8767,  
3041 ["wreath"] = 8768,  
3042 ["VerticalTilde"] = 8768,  
3043 ["wr"] = 8768,  
3044 ["nsim"] = 8769,  
3045 ["NotTilde"] = 8769,  
3046 ["esim"] = 8770,  
3047 ["EqualTilde"] = 8770,  
3048 ["eqsim"] = 8770,  
3049 ["sime"] = 8771,  
3050 ["TildeEqual"] = 8771,  
3051 ["simeq"] = 8771,



3052 ["nsime"] = 8772,  
3053 ["nsimeq"] = 8772,  
3054 ["NotTildeEqual"] = 8772,  
3055 ["cong"] = 8773,  
3056 ["TildeFullEqual"] = 8773,  
3057 ["simne"] = 8774,  
3058 ["ncong"] = 8775,  
3059 ["NotTildeFullEqual"] = 8775,  
3060 ["asymp"] = 8776,  
3061 ["ap"] = 8776,  
3062 ["TildeTilde"] = 8776,  
3063 ["approx"] = 8776,  
3064 ["thkap"] = 8776,  
3065 ["thickapprox"] = 8776,  
3066 ["nap"] = 8777,  
3067 ["NotTildeTilde"] = 8777,  
3068 ["naprox"] = 8777,  
3069 ["ape"] = 8778,  
3070 ["approxpeq"] = 8778,  
3071 ["apid"] = 8779,  
3072 ["bcong"] = 8780,  
3073 ["backcong"] = 8780,  
3074 ["asympeq"] = 8781,  
3075 ["CupCap"] = 8781,  
3076 ["bump"] = 8782,  
3077 ["HumpDownHump"] = 8782,  
3078 ["Bumpeq"] = 8782,  
3079 ["bumpe"] = 8783,  
3080 ["HumpEqual"] = 8783,  
3081 ["bumpeq"] = 8783,  
3082 ["esdot"] = 8784,  
3083 ["DotEqual"] = 8784,  
3084 ["doteq"] = 8784,  
3085 ["eDot"] = 8785,  
3086 ["doteqdot"] = 8785,  
3087 ["efDot"] = 8786,  
3088 ["fallingdotseq"] = 8786,  
3089 ["erDot"] = 8787,  
3090 ["risingdotseq"] = 8787,  
3091 ["colone"] = 8788,  
3092 ["coloneq"] = 8788,  
3093 ["Assign"] = 8788,  
3094 ["ecolon"] = 8789,  
3095 ["eqcolon"] = 8789,  
3096 ["ecir"] = 8790,  
3097 ["eqcirc"] = 8790,  
3098 ["cire"] = 8791,

3099 ["circeq"] = 8791,  
3100 ["wedgeq"] = 8793,  
3101 ["veeeq"] = 8794,  
3102 ["trie"] = 8796,  
3103 ["triangleq"] = 8796,  
3104 ["equest"] = 8799,  
3105 ["questeq"] = 8799,  
3106 ["ne"] = 8800,  
3107 ["NotEqual"] = 8800,  
3108 ["equiv"] = 8801,  
3109 ["Congruent"] = 8801,  
3110 ["nequiv"] = 8802,  
3111 ["NotCongruent"] = 8802,  
3112 ["le"] = 8804,  
3113 ["leq"] = 8804,  
3114 ["ge"] = 8805,  
3115 ["GreaterEqual"] = 8805,  
3116 ["geq"] = 8805,  
3117 ["lE"] = 8806,  
3118 ["LessFullEqual"] = 8806,  
3119 ["leqq"] = 8806,  
3120 ["gE"] = 8807,  
3121 ["GreaterFullEqual"] = 8807,  
3122 ["geqq"] = 8807,  
3123 ["lnE"] = 8808,  
3124 ["lneqq"] = 8808,  
3125 ["gnE"] = 8809,  
3126 ["gneqq"] = 8809,  
3127 ["Lt"] = 8810,  
3128 ["NestedLessLess"] = 8810,  
3129 ["ll"] = 8810,  
3130 ["Gt"] = 8811,  
3131 ["NestedGreaterGreater"] = 8811,  
3132 ["gg"] = 8811,  
3133 ["twixt"] = 8812,  
3134 ["between"] = 8812,  
3135 ["NotCupCap"] = 8813,  
3136 ["nlt"] = 8814,  
3137 ["NotLess"] = 8814,  
3138 ["nless"] = 8814,  
3139 ["ngt"] = 8815,  
3140 ["NotGreater"] = 8815,  
3141 ["ngtr"] = 8815,  
3142 ["nle"] = 8816,  
3143 ["NotLessEqual"] = 8816,  
3144 ["nleq"] = 8816,  
3145 ["nge"] = 8817,

3146 ["NotGreaterEqual"] = 8817,  
 3147 ["ngeq"] = 8817,  
 3148 ["lsim"] = 8818,  
 3149 ["LessTilde"] = 8818,  
 3150 ["lesssim"] = 8818,  
 3151 ["gsim"] = 8819,  
 3152 ["gtrsim"] = 8819,  
 3153 ["GreaterTilde"] = 8819,  
 3154 ["nlsim"] = 8820,  
 3155 ["NotLessTilde"] = 8820,  
 3156 ["ngsim"] = 8821,  
 3157 ["NotGreaterTilde"] = 8821,  
 3158 ["lg"] = 8822,  
 3159 ["lessgtr"] = 8822,  
 3160 ["LessGreater"] = 8822,  
 3161 ["gl"] = 8823,  
 3162 ["gtrless"] = 8823,  
 3163 ["GreaterLess"] = 8823,  
 3164 ["ntlg"] = 8824,  
 3165 ["NotLessGreater"] = 8824,  
 3166 ["ntgl"] = 8825,  
 3167 ["NotGreaterLess"] = 8825,  
 3168 ["pr"] = 8826,  
 3169 ["Precedes"] = 8826,  
 3170 ["prec"] = 8826,  
 3171 ["sc"] = 8827,  
 3172 ["Succeeds"] = 8827,  
 3173 ["succ"] = 8827,  
 3174 ["prcue"] = 8828,  
 3175 ["PrecedesSlantEqual"] = 8828,  
 3176 ["preccurlyeq"] = 8828,  
 3177 ["sccue"] = 8829,  
 3178 ["SucceedsSlantEqual"] = 8829,  
 3179 ["succcurlyeq"] = 8829,  
 3180 ["prsim"] = 8830,  
 3181 ["precsim"] = 8830,  
 3182 ["PrecedesTilde"] = 8830,  
 3183 ["scsim"] = 8831,  
 3184 ["succsim"] = 8831,  
 3185 ["SucceedsTilde"] = 8831,  
 3186 ["npr"] = 8832,  
 3187 ["nprec"] = 8832,  
 3188 ["NotPrecedes"] = 8832,  
 3189 ["nsc"] = 8833,  
 3190 ["nsucc"] = 8833,  
 3191 ["NotSucceeds"] = 8833,  
 3192 ["sub"] = 8834,

3193 ["subset"] = 8834,  
3194 ["sup"] = 8835,  
3195 ["supset"] = 8835,  
3196 ["Superset"] = 8835,  
3197 ["nsub"] = 8836,  
3198 ["nsup"] = 8837,  
3199 ["sube"] = 8838,  
3200 ["SubsetEqual"] = 8838,  
3201 ["subseteq"] = 8838,  
3202 ["supe"] = 8839,  
3203 ["supseteq"] = 8839,  
3204 ["SupersetEqual"] = 8839,  
3205 ["nsube"] = 8840,  
3206 ["nsubseteq"] = 8840,  
3207 ["NotSubsetEqual"] = 8840,  
3208 ["nsupe"] = 8841,  
3209 ["nsupseteq"] = 8841,  
3210 ["NotSupersetEqual"] = 8841,  
3211 ["subne"] = 8842,  
3212 ["subsetneq"] = 8842,  
3213 ["supne"] = 8843,  
3214 ["supsetneq"] = 8843,  
3215 ["cupdot"] = 8845,  
3216 ["uplus"] = 8846,  
3217 ["UnionPlus"] = 8846,  
3218 ["sqsub"] = 8847,  
3219 ["SquareSubset"] = 8847,  
3220 ["sqsubset"] = 8847,  
3221 ["sqsup"] = 8848,  
3222 ["SquareSuperset"] = 8848,  
3223 ["sqsupset"] = 8848,  
3224 ["sqsube"] = 8849,  
3225 ["SquareSubsetEqual"] = 8849,  
3226 ["sqsubseteq"] = 8849,  
3227 ["sqsupe"] = 8850,  
3228 ["SquareSupersetEqual"] = 8850,  
3229 ["sqsupseteq"] = 8850,  
3230 ["sqcap"] = 8851,  
3231 ["SquareIntersection"] = 8851,  
3232 ["sqcup"] = 8852,  
3233 ["SquareUnion"] = 8852,  
3234 ["oplus"] = 8853,  
3235 ["CirclePlus"] = 8853,  
3236 ["ominus"] = 8854,  
3237 ["CircleMinus"] = 8854,  
3238 ["otimes"] = 8855,  
3239 ["CircleTimes"] = 8855,

3240 ["osol"] = 8856,  
 3241 ["odot"] = 8857,  
 3242 ["CircleDot"] = 8857,  
 3243 ["ocir"] = 8858,  
 3244 ["circledcirc"] = 8858,  
 3245 ["oast"] = 8859,  
 3246 ["circledast"] = 8859,  
 3247 ["odash"] = 8861,  
 3248 ["circleddash"] = 8861,  
 3249 ["plusb"] = 8862,  
 3250 ["boxplus"] = 8862,  
 3251 ["minusb"] = 8863,  
 3252 ["boxminus"] = 8863,  
 3253 ["timesb"] = 8864,  
 3254 ["boxtimes"] = 8864,  
 3255 ["sdotb"] = 8865,  
 3256 ["dotsquare"] = 8865,  
 3257 ["vdash"] = 8866,  
 3258 ["RightTee"] = 8866,  
 3259 ["dashv"] = 8867,  
 3260 ["LeftTee"] = 8867,  
 3261 ["top"] = 8868,  
 3262 ["DownTee"] = 8868,  
 3263 ["bottom"] = 8869,  
 3264 ["bot"] = 8869,  
 3265 ["perp"] = 8869,  
 3266 ["UpTee"] = 8869,  
 3267 ["models"] = 8871,  
 3268 ["vDash"] = 8872,  
 3269 ["DoubleRightTee"] = 8872,  
 3270 ["Vdash"] = 8873,  
 3271 ["Vvdash"] = 8874,  
 3272 ["VDash"] = 8875,  
 3273 ["nvdash"] = 8876,  
 3274 ["nvDash"] = 8877,  
 3275 ["nVdash"] = 8878,  
 3276 ["nVDash"] = 8879,  
 3277 ["prurel"] = 8880,  
 3278 ["vltri"] = 8882,  
 3279 ["vartriangleleft"] = 8882,  
 3280 ["LeftTriangle"] = 8882,  
 3281 ["vrtri"] = 8883,  
 3282 ["vartriangleright"] = 8883,  
 3283 ["RightTriangle"] = 8883,  
 3284 ["ltrie"] = 8884,  
 3285 ["trianglelefteq"] = 8884,  
 3286 ["LeftTriangleEqual"] = 8884,

3287 ["rtrie"] = 8885,  
 3288 ["trianglerighteq"] = 8885,  
 3289 ["RightTriangleEqual"] = 8885,  
 3290 ["origof"] = 8886,  
 3291 ["imof"] = 8887,  
 3292 ["mumap"] = 8888,  
 3293 ["multimap"] = 8888,  
 3294 ["hercon"] = 8889,  
 3295 ["intcal"] = 8890,  
 3296 ["intercal"] = 8890,  
 3297 ["veebar"] = 8891,  
 3298 ["barvee"] = 8893,  
 3299 ["angrtvb"] = 8894,  
 3300 ["lrtri"] = 8895,  
 3301 ["xwedge"] = 8896,  
 3302 ["Wedge"] = 8896,  
 3303 ["bigwedge"] = 8896,  
 3304 ["xvee"] = 8897,  
 3305 ["Vee"] = 8897,  
 3306 ["bigvee"] = 8897,  
 3307 ["xcap"] = 8898,  
 3308 ["Intersection"] = 8898,  
 3309 ["bigcap"] = 8898,  
 3310 ["xcup"] = 8899,  
 3311 ["Union"] = 8899,  
 3312 ["bigcup"] = 8899,  
 3313 ["diam"] = 8900,  
 3314 ["diamond"] = 8900,  
 3315 ["Diamond"] = 8900,  
 3316 ["sdot"] = 8901,  
 3317 ["sstarf"] = 8902,  
 3318 ["Star"] = 8902,  
 3319 ["divonx"] = 8903,  
 3320 ["divideontimes"] = 8903,  
 3321 ["bowtie"] = 8904,  
 3322 ["ltimes"] = 8905,  
 3323 ["rtimes"] = 8906,  
 3324 ["lthree"] = 8907,  
 3325 ["leftthreetimes"] = 8907,  
 3326 ["rthree"] = 8908,  
 3327 ["rightthreetimes"] = 8908,  
 3328 ["bsime"] = 8909,  
 3329 ["backsimeq"] = 8909,  
 3330 ["cuvee"] = 8910,  
 3331 ["curlyvee"] = 8910,  
 3332 ["cuwed"] = 8911,  
 3333 ["curlywedge"] = 8911,

3334 ["Sub"] = 8912,  
3335 ["Subset"] = 8912,  
3336 ["Sup"] = 8913,  
3337 ["Supset"] = 8913,  
3338 ["Cap"] = 8914,  
3339 ["Cup"] = 8915,  
3340 ["fork"] = 8916,  
3341 ["pitchfork"] = 8916,  
3342 ["epar"] = 8917,  
3343 ["ltdot"] = 8918,  
3344 ["lessdot"] = 8918,  
3345 ["gtdot"] = 8919,  
3346 ["gtrdot"] = 8919,  
3347 ["Ll"] = 8920,  
3348 ["Gg"] = 8921,  
3349 ["ggg"] = 8921,  
3350 ["leg"] = 8922,  
3351 ["LessEqualGreater"] = 8922,  
3352 ["lesseqgtr"] = 8922,  
3353 ["gel"] = 8923,  
3354 ["gtreqless"] = 8923,  
3355 ["GreaterEqualLess"] = 8923,  
3356 ["cuepr"] = 8926,  
3357 ["curlyeqprec"] = 8926,  
3358 ["cuesc"] = 8927,  
3359 ["curlyeqsucc"] = 8927,  
3360 ["nprcue"] = 8928,  
3361 ["NotPrecedesSlantEqual"] = 8928,  
3362 ["nscue"] = 8929,  
3363 ["NotSucceedsSlantEqual"] = 8929,  
3364 ["nsqsube"] = 8930,  
3365 ["NotSquareSubsetEqual"] = 8930,  
3366 ["nsqsupe"] = 8931,  
3367 ["NotSquareSupersetEqual"] = 8931,  
3368 ["lnsim"] = 8934,  
3369 ["gnsim"] = 8935,  
3370 ["prnsim"] = 8936,  
3371 ["precnsim"] = 8936,  
3372 ["scnsim"] = 8937,  
3373 ["succnsim"] = 8937,  
3374 ["nltri"] = 8938,  
3375 ["ntriangleleft"] = 8938,  
3376 ["NotLeftTriangle"] = 8938,  
3377 ["nrtri"] = 8939,  
3378 ["ntriangleright"] = 8939,  
3379 ["NotRightTriangle"] = 8939,  
3380 ["nltrie"] = 8940,

3381 ["ntrianglelefteq"] = 8940,  
 3382 ["NotLeftTriangleEqual"] = 8940,  
 3383 ["nrtrie"] = 8941,  
 3384 ["ntrianglerighteq"] = 8941,  
 3385 ["NotRightTriangleEqual"] = 8941,  
 3386 ["vellip"] = 8942,  
 3387 ["ctdot"] = 8943,  
 3388 ["utdot"] = 8944,  
 3389 ["dtdot"] = 8945,  
 3390 ["disin"] = 8946,  
 3391 ["isinsv"] = 8947,  
 3392 ["isins"] = 8948,  
 3393 ["isindot"] = 8949,  
 3394 ["notinvc"] = 8950,  
 3395 ["notinvb"] = 8951,  
 3396 ["isinE"] = 8953,  
 3397 ["nisd"] = 8954,  
 3398 ["xnis"] = 8955,  
 3399 ["nis"] = 8956,  
 3400 ["notnivc"] = 8957,  
 3401 ["notnivb"] = 8958,  
 3402 ["barwed"] = 8965,  
 3403 ["barwedge"] = 8965,  
 3404 ["Barwed"] = 8966,  
 3405 ["doublebarwedge"] = 8966,  
 3406 ["lceil"] = 8968,  
 3407 ["LeftCeiling"] = 8968,  
 3408 ["rceil"] = 8969,  
 3409 ["RightCeiling"] = 8969,  
 3410 ["lfloor"] = 8970,  
 3411 ["LeftFloor"] = 8970,  
 3412 ["rfloor"] = 8971,  
 3413 ["RightFloor"] = 8971,  
 3414 ["drcrop"] = 8972,  
 3415 ["dlcrop"] = 8973,  
 3416 ["urcrop"] = 8974,  
 3417 ["ulcrop"] = 8975,  
 3418 ["bnot"] = 8976,  
 3419 ["proflin"] = 8978,  
 3420 ["profsurf"] = 8979,  
 3421 ["telrec"] = 8981,  
 3422 ["target"] = 8982,  
 3423 ["ulcorn"] = 8988,  
 3424 ["ulcorner"] = 8988,  
 3425 ["urcorn"] = 8989,  
 3426 ["urcorner"] = 8989,  
 3427 ["dlcorn"] = 8990,



3428 ["llcorner"] = 8990,  
3429 ["drcorn"] = 8991,  
3430 ["lrcorner"] = 8991,  
3431 ["frown"] = 8994,  
3432 ["sfrown"] = 8994,  
3433 ["smile"] = 8995,  
3434 ["ssmile"] = 8995,  
3435 ["cylcty"] = 9005,  
3436 ["profalar"] = 9006,  
3437 ["topbot"] = 9014,  
3438 ["ovbar"] = 9021,  
3439 ["solbar"] = 9023,  
3440 ["angzarr"] = 9084,  
3441 ["lmoust"] = 9136,  
3442 ["lmoustache"] = 9136,  
3443 ["rmoust"] = 9137,  
3444 ["rmoustache"] = 9137,  
3445 ["tbrk"] = 9140,  
3446 ["OverBracket"] = 9140,  
3447 ["bbrk"] = 9141,  
3448 ["UnderBracket"] = 9141,  
3449 ["bbrktbrk"] = 9142,  
3450 ["OverParenthesis"] = 9180,  
3451 ["UnderParenthesis"] = 9181,  
3452 ["OverBrace"] = 9182,  
3453 ["UnderBrace"] = 9183,  
3454 ["trpezium"] = 9186,  
3455 ["elinters"] = 9191,  
3456 ["blank"] = 9251,  
3457 ["oS"] = 9416,  
3458 ["circledS"] = 9416,  
3459 ["boxh"] = 9472,  
3460 ["HorizontalLine"] = 9472,  
3461 ["boxv"] = 9474,  
3462 ["boxdr"] = 9484,  
3463 ["boxdl"] = 9488,  
3464 ["boxur"] = 9492,  
3465 ["boxul"] = 9496,  
3466 ["boxvr"] = 9500,  
3467 ["boxvl"] = 9508,  
3468 ["boxhd"] = 9516,  
3469 ["boxhu"] = 9524,  
3470 ["boxvh"] = 9532,  
3471 ["boxH"] = 9552,  
3472 ["boxV"] = 9553,  
3473 ["boxdR"] = 9554,  
3474 ["boxDr"] = 9555,

3475 ["boxDR"] = 9556,  
 3476 ["boxdL"] = 9557,  
 3477 ["boxDl"] = 9558,  
 3478 ["boxDL"] = 9559,  
 3479 ["boxuR"] = 9560,  
 3480 ["boxUr"] = 9561,  
 3481 ["boxUR"] = 9562,  
 3482 ["boxuL"] = 9563,  
 3483 ["boxUl"] = 9564,  
 3484 ["boxUL"] = 9565,  
 3485 ["boxvR"] = 9566,  
 3486 ["boxVr"] = 9567,  
 3487 ["boxVR"] = 9568,  
 3488 ["boxvL"] = 9569,  
 3489 ["boxVl"] = 9570,  
 3490 ["boxVL"] = 9571,  
 3491 ["boxHd"] = 9572,  
 3492 ["boxhD"] = 9573,  
 3493 ["boxHD"] = 9574,  
 3494 ["boxHu"] = 9575,  
 3495 ["boxhU"] = 9576,  
 3496 ["boxHU"] = 9577,  
 3497 ["boxvH"] = 9578,  
 3498 ["boxVh"] = 9579,  
 3499 ["boxVH"] = 9580,  
 3500 ["uhblk"] = 9600,  
 3501 ["lhblk"] = 9604,  
 3502 ["block"] = 9608,  
 3503 ["blk14"] = 9617,  
 3504 ["blk12"] = 9618,  
 3505 ["blk34"] = 9619,  
 3506 ["squ"] = 9633,  
 3507 ["square"] = 9633,  
 3508 ["Square"] = 9633,  
 3509 ["squf"] = 9642,  
 3510 ["squarf"] = 9642,  
 3511 ["blacksquare"] = 9642,  
 3512 ["FilledVerySmallSquare"] = 9642,  
 3513 ["EmptyVerySmallSquare"] = 9643,  
 3514 ["rect"] = 9645,  
 3515 ["marker"] = 9646,  
 3516 ["fltns"] = 9649,  
 3517 ["xutri"] = 9651,  
 3518 ["bigtriangleup"] = 9651,  
 3519 ["utrif"] = 9652,  
 3520 ["blacktriangle"] = 9652,  
 3521 ["utri"] = 9653,

3522 ["triangle"] = 9653,  
3523 ["rtrif"] = 9656,  
3524 ["blacktriangleright"] = 9656,  
3525 ["rtri"] = 9657,  
3526 ["triangleright"] = 9657,  
3527 ["xdtri"] = 9661,  
3528 ["bigtriangledown"] = 9661,  
3529 ["dtrif"] = 9662,  
3530 ["blacktriangledown"] = 9662,  
3531 ["dtri"] = 9663,  
3532 ["triangledown"] = 9663,  
3533 ["ltrif"] = 9666,  
3534 ["blacktriangleleft"] = 9666,  
3535 ["ltrif"] = 9667,  
3536 ["triangleleft"] = 9667,  
3537 ["loz"] = 9674,  
3538 ["lozenge"] = 9674,  
3539 ["cir"] = 9675,  
3540 ["tridot"] = 9708,  
3541 ["xcirc"] = 9711,  
3542 ["bigcirc"] = 9711,  
3543 ["ultri"] = 9720,  
3544 ["urtri"] = 9721,  
3545 ["lltri"] = 9722,  
3546 ["EmptySmallSquare"] = 9723,  
3547 ["FilledSmallSquare"] = 9724,  
3548 ["starf"] = 9733,  
3549 ["bigstar"] = 9733,  
3550 ["star"] = 9734,  
3551 ["phone"] = 9742,  
3552 ["female"] = 9792,  
3553 ["male"] = 9794,  
3554 ["spades"] = 9824,  
3555 ["spadesuit"] = 9824,  
3556 ["clubs"] = 9827,  
3557 ["clubsuit"] = 9827,  
3558 ["hearts"] = 9829,  
3559 ["heartsuit"] = 9829,  
3560 ["diams"] = 9830,  
3561 ["diamondsuit"] = 9830,  
3562 ["sung"] = 9834,  
3563 ["flat"] = 9837,  
3564 ["natur"] = 9838,  
3565 ["natural"] = 9838,  
3566 ["sharp"] = 9839,  
3567 ["check"] = 10003,  
3568 ["checkmark"] = 10003,

3569 ["cross"] = 10007,  
 3570 ["malt"] = 10016,  
 3571 ["maltese"] = 10016,  
 3572 ["sext"] = 10038,  
 3573 ["VerticalSeparator"] = 10072,  
 3574 ["lbrk"] = 10098,  
 3575 ["rbrk"] = 10099,  
 3576 ["lobrk"] = 10214,  
 3577 ["LeftDoubleBracket"] = 10214,  
 3578 ["robrk"] = 10215,  
 3579 ["RightDoubleBracket"] = 10215,  
 3580 ["lang"] = 10216,  
 3581 ["LeftAngleBracket"] = 10216,  
 3582 ["langle"] = 10216,  
 3583 ["rang"] = 10217,  
 3584 ["RightAngleBracket"] = 10217,  
 3585 ["rangle"] = 10217,  
 3586 ["Lang"] = 10218,  
 3587 ["Rang"] = 10219,  
 3588 ["loang"] = 10220,  
 3589 ["roang"] = 10221,  
 3590 ["xlarr"] = 10229,  
 3591 ["longleftarrow"] = 10229,  
 3592 ["LongLeftArrow"] = 10229,  
 3593 ["xrarr"] = 10230,  
 3594 ["longrightarrow"] = 10230,  
 3595 ["LongRightArrow"] = 10230,  
 3596 ["xharr"] = 10231,  
 3597 ["longlefttrightarrow"] = 10231,  
 3598 ["LongLeftRightArrow"] = 10231,  
 3599 ["xlArr"] = 10232,  
 3600 ["Longleftarrow"] = 10232,  
 3601 ["DoubleLongLeftArrow"] = 10232,  
 3602 ["xrArr"] = 10233,  
 3603 ["Longrightarrow"] = 10233,  
 3604 ["DoubleLongRightArrow"] = 10233,  
 3605 ["xhArr"] = 10234,  
 3606 ["Longlefttrightarrow"] = 10234,  
 3607 ["DoubleLongLeftRightArrow"] = 10234,  
 3608 ["xmap"] = 10236,  
 3609 ["longmapsto"] = 10236,  
 3610 ["dzigrarr"] = 10239,  
 3611 ["nvlArr"] = 10498,  
 3612 ["nvrArr"] = 10499,  
 3613 ["nvHarr"] = 10500,  
 3614 ["Map"] = 10501,  
 3615 ["lbarr"] = 10508,

3616 ["rbarr"] = 10509,  
 3617 ["bkarow"] = 10509,  
 3618 ["lBarr"] = 10510,  
 3619 ["rBarr"] = 10511,  
 3620 ["dbkarow"] = 10511,  
 3621 ["RBarr"] = 10512,  
 3622 ["drbkarow"] = 10512,  
 3623 ["DDotrahd"] = 10513,  
 3624 ["UpArrowBar"] = 10514,  
 3625 ["DownArrowBar"] = 10515,  
 3626 ["Rarrtl"] = 10518,  
 3627 ["latail"] = 10521,  
 3628 ["ratail"] = 10522,  
 3629 ["lAtail"] = 10523,  
 3630 ["rAtail"] = 10524,  
 3631 ["larrfs"] = 10525,  
 3632 ["rarrfs"] = 10526,  
 3633 ["larrbfs"] = 10527,  
 3634 ["rarrbfs"] = 10528,  
 3635 ["nwarhk"] = 10531,  
 3636 ["nearhk"] = 10532,  
 3637 ["searhk"] = 10533,  
 3638 ["hksearow"] = 10533,  
 3639 ["swarhk"] = 10534,  
 3640 ["hkswarow"] = 10534,  
 3641 ["nwnear"] = 10535,  
 3642 ["nesear"] = 10536,  
 3643 ["toea"] = 10536,  
 3644 ["seswar"] = 10537,  
 3645 ["tosa"] = 10537,  
 3646 ["swnwar"] = 10538,  
 3647 ["rarrc"] = 10547,  
 3648 ["cudarr"] = 10549,  
 3649 ["ldca"] = 10550,  
 3650 ["rdca"] = 10551,  
 3651 ["cudarrl"] = 10552,  
 3652 ["larrpl"] = 10553,  
 3653 ["curarrm"] = 10556,  
 3654 ["cularrp"] = 10557,  
 3655 ["rarrpl"] = 10565,  
 3656 ["harrcir"] = 10568,  
 3657 ["Uarrocir"] = 10569,  
 3658 ["lurdshar"] = 10570,  
 3659 ["ldrushar"] = 10571,  
 3660 ["LeftRightVector"] = 10574,  
 3661 ["RightUpDownVector"] = 10575,  
 3662 ["DownLeftRightVector"] = 10576,

3663 ["LeftUpDownVector"] = 10577,  
 3664 ["LeftVectorBar"] = 10578,  
 3665 ["RightVectorBar"] = 10579,  
 3666 ["RightUpVectorBar"] = 10580,  
 3667 ["RightDownVectorBar"] = 10581,  
 3668 ["DownLeftVectorBar"] = 10582,  
 3669 ["DownRightVectorBar"] = 10583,  
 3670 ["LeftUpVectorBar"] = 10584,  
 3671 ["LeftDownVectorBar"] = 10585,  
 3672 ["LeftTeeVector"] = 10586,  
 3673 ["RightTeeVector"] = 10587,  
 3674 ["RightUpTeeVector"] = 10588,  
 3675 ["RightDownTeeVector"] = 10589,  
 3676 ["DownLeftTeeVector"] = 10590,  
 3677 ["DownRightTeeVector"] = 10591,  
 3678 ["LeftUpTeeVector"] = 10592,  
 3679 ["LeftDownTeeVector"] = 10593,  
 3680 ["lHar"] = 10594,  
 3681 ["uHar"] = 10595,  
 3682 ["rHar"] = 10596,  
 3683 ["dHar"] = 10597,  
 3684 ["luruhar"] = 10598,  
 3685 ["ldrdhar"] = 10599,  
 3686 ["ruluhar"] = 10600,  
 3687 ["rdldhar"] = 10601,  
 3688 ["lharul"] = 10602,  
 3689 ["llhard"] = 10603,  
 3690 ["rharul"] = 10604,  
 3691 ["lrhard"] = 10605,  
 3692 ["udhar"] = 10606,  
 3693 ["UpEquilibrium"] = 10606,  
 3694 ["duhar"] = 10607,  
 3695 ["ReverseUpEquilibrium"] = 10607,  
 3696 ["RoundImplies"] = 10608,  
 3697 ["erarr"] = 10609,  
 3698 ["simrarr"] = 10610,  
 3699 ["larrsim"] = 10611,  
 3700 ["rarrsim"] = 10612,  
 3701 ["rarrap"] = 10613,  
 3702 ["ltlarr"] = 10614,  
 3703 ["gtrarr"] = 10616,  
 3704 ["subrarr"] = 10617,  
 3705 ["suplarr"] = 10619,  
 3706 ["lfisht"] = 10620,  
 3707 ["rfisht"] = 10621,  
 3708 ["ufisht"] = 10622,  
 3709 ["dfisht"] = 10623,

3710 ["lopar"] = 10629,  
3711 ["ropar"] = 10630,  
3712 ["lbrke"] = 10635,  
3713 ["rbrke"] = 10636,  
3714 ["lbrkslu"] = 10637,  
3715 ["rbrksld"] = 10638,  
3716 ["lbrksld"] = 10639,  
3717 ["rbrkslu"] = 10640,  
3718 ["langd"] = 10641,  
3719 ["rangd"] = 10642,  
3720 ["lparlt"] = 10643,  
3721 ["rpargt"] = 10644,  
3722 ["gtlPar"] = 10645,  
3723 ["ltrPar"] = 10646,  
3724 ["vzigzag"] = 10650,  
3725 ["vangrt"] = 10652,  
3726 ["angrtvbd"] = 10653,  
3727 ["ange"] = 10660,  
3728 ["range"] = 10661,  
3729 ["dwangle"] = 10662,  
3730 ["uwangle"] = 10663,  
3731 ["angmsdaa"] = 10664,  
3732 ["angmsdab"] = 10665,  
3733 ["angmsdac"] = 10666,  
3734 ["angmsdad"] = 10667,  
3735 ["angmsdae"] = 10668,  
3736 ["angmsdaf"] = 10669,  
3737 ["angmsdag"] = 10670,  
3738 ["angmsdah"] = 10671,  
3739 ["bemptyv"] = 10672,  
3740 ["demptyv"] = 10673,  
3741 ["cemptyv"] = 10674,  
3742 ["raemptyv"] = 10675,  
3743 ["laemptyv"] = 10676,  
3744 ["ohbar"] = 10677,  
3745 ["omid"] = 10678,  
3746 ["opar"] = 10679,  
3747 ["operp"] = 10681,  
3748 ["olcross"] = 10683,  
3749 ["odsold"] = 10684,  
3750 ["olcir"] = 10686,  
3751 ["ofcir"] = 10687,  
3752 ["olt"] = 10688,  
3753 ["ogt"] = 10689,  
3754 ["cirscir"] = 10690,  
3755 ["cirE"] = 10691,  
3756 ["solb"] = 10692,

3757 ["bsolb"] = 10693,  
3758 ["boxbox"] = 10697,  
3759 ["trisb"] = 10701,  
3760 ["rtriltri"] = 10702,  
3761 ["LeftTriangleBar"] = 10703,  
3762 ["RightTriangleBar"] = 10704,  
3763 ["race"] = 10714,  
3764 ["iinfin"] = 10716,  
3765 ["infintie"] = 10717,  
3766 ["nvinfin"] = 10718,  
3767 ["eparsl"] = 10723,  
3768 ["smeparsl"] = 10724,  
3769 ["eqvparsl"] = 10725,  
3770 ["lozf"] = 10731,  
3771 ["blacklozenge"] = 10731,  
3772 ["RuleDelayed"] = 10740,  
3773 ["dsol"] = 10742,  
3774 ["xodot"] = 10752,  
3775 ["bigodot"] = 10752,  
3776 ["xoplus"] = 10753,  
3777 ["bigoplus"] = 10753,  
3778 ["xotime"] = 10754,  
3779 ["bigotimes"] = 10754,  
3780 ["xuplus"] = 10756,  
3781 ["biguplus"] = 10756,  
3782 ["xsqcup"] = 10758,  
3783 ["bigsqcup"] = 10758,  
3784 ["qint"] = 10764,  
3785 ["iiiint"] = 10764,  
3786 ["fpartint"] = 10765,  
3787 ["cirfnint"] = 10768,  
3788 ["awint"] = 10769,  
3789 ["rppolint"] = 10770,  
3790 ["scpolint"] = 10771,  
3791 ["npolint"] = 10772,  
3792 ["pointint"] = 10773,  
3793 ["quatint"] = 10774,  
3794 ["intlarhk"] = 10775,  
3795 ["pluscir"] = 10786,  
3796 ["plusacir"] = 10787,  
3797 ["simplus"] = 10788,  
3798 ["plusdu"] = 10789,  
3799 ["plussim"] = 10790,  
3800 ["plustwo"] = 10791,  
3801 ["mcomma"] = 10793,  
3802 ["minusdu"] = 10794,  
3803 ["loplus"] = 10797,



3804 ["roplus"] = 10798,  
3805 ["Cross"] = 10799,  
3806 ["timesd"] = 10800,  
3807 ["timesbar"] = 10801,  
3808 ["smashp"] = 10803,  
3809 ["lotimes"] = 10804,  
3810 ["rotimes"] = 10805,  
3811 ["otimesas"] = 10806,  
3812 ["Otimes"] = 10807,  
3813 ["odiv"] = 10808,  
3814 ["triplus"] = 10809,  
3815 ["triminus"] = 10810,  
3816 ["tritime"] = 10811,  
3817 ["iproduct"] = 10812,  
3818 ["intprod"] = 10812,  
3819 ["amalg"] = 10815,  
3820 ["capdot"] = 10816,  
3821 ["ncup"] = 10818,  
3822 ["ncap"] = 10819,  
3823 ["capand"] = 10820,  
3824 ["cupor"] = 10821,  
3825 ["cupcap"] = 10822,  
3826 ["capcup"] = 10823,  
3827 ["cupbrcap"] = 10824,  
3828 ["capbrcup"] = 10825,  
3829 ["cupcup"] = 10826,  
3830 ["capcap"] = 10827,  
3831 ["ccups"] = 10828,  
3832 ["ccaps"] = 10829,  
3833 ["ccupssm"] = 10832,  
3834 ["And"] = 10835,  
3835 ["Or"] = 10836,  
3836 ["andand"] = 10837,  
3837 ["oror"] = 10838,  
3838 ["orslope"] = 10839,  
3839 ["andslope"] = 10840,  
3840 ["andv"] = 10842,  
3841 ["orv"] = 10843,  
3842 ["andd"] = 10844,  
3843 ["ord"] = 10845,  
3844 ["wedbar"] = 10847,  
3845 ["sdote"] = 10854,  
3846 ["simdot"] = 10858,  
3847 ["congdot"] = 10861,  
3848 ["easter"] = 10862,  
3849 ["apacir"] = 10863,  
3850 ["apE"] = 10864,

3851 ["eplus"] = 10865,  
 3852 ["pluse"] = 10866,  
 3853 ["Esim"] = 10867,  
 3854 ["Colone"] = 10868,  
 3855 ["Equal"] = 10869,  
 3856 ["eDDot"] = 10871,  
 3857 ["ddotseq"] = 10871,  
 3858 ["equivDD"] = 10872,  
 3859 ["ltcir"] = 10873,  
 3860 ["gtcir"] = 10874,  
 3861 ["ltquest"] = 10875,  
 3862 ["gtquest"] = 10876,  
 3863 ["les"] = 10877,  
 3864 ["LessSlantEqual"] = 10877,  
 3865 ["leqslant"] = 10877,  
 3866 ["ges"] = 10878,  
 3867 ["GreaterSlantEqual"] = 10878,  
 3868 ["geqslant"] = 10878,  
 3869 ["lesdot"] = 10879,  
 3870 ["gesdot"] = 10880,  
 3871 ["lesdoto"] = 10881,  
 3872 ["gesdoto"] = 10882,  
 3873 ["lesdotor"] = 10883,  
 3874 ["gesdoto1"] = 10884,  
 3875 ["lap"] = 10885,  
 3876 ["lessapprox"] = 10885,  
 3877 ["gap"] = 10886,  
 3878 ["gtrapprox"] = 10886,  
 3879 ["lne"] = 10887,  
 3880 ["lneq"] = 10887,  
 3881 ["gne"] = 10888,  
 3882 ["gneq"] = 10888,  
 3883 ["lnap"] = 10889,  
 3884 ["lnapprox"] = 10889,  
 3885 ["gnap"] = 10890,  
 3886 ["gnapprox"] = 10890,  
 3887 ["lEg"] = 10891,  
 3888 ["lesseqqgtr"] = 10891,  
 3889 ["gEl"] = 10892,  
 3890 ["gtreqqless"] = 10892,  
 3891 ["lsime"] = 10893,  
 3892 ["gsime"] = 10894,  
 3893 ["lsimg"] = 10895,  
 3894 ["gsiml"] = 10896,  
 3895 ["lgE"] = 10897,  
 3896 ["glE"] = 10898,  
 3897 ["lesges"] = 10899,

3898 ["gesles"] = 10900,  
3899 ["els"] = 10901,  
3900 ["eqslantless"] = 10901,  
3901 ["egs"] = 10902,  
3902 ["eqslantgtr"] = 10902,  
3903 ["elsdot"] = 10903,  
3904 ["egsdot"] = 10904,  
3905 ["el"] = 10905,  
3906 ["eg"] = 10906,  
3907 ["siml"] = 10909,  
3908 ["simg"] = 10910,  
3909 ["simLE"] = 10911,  
3910 ["simGE"] = 10912,  
3911 ["LessLess"] = 10913,  
3912 ["GreaterGreater"] = 10914,  
3913 ["glj"] = 10916,  
3914 ["gla"] = 10917,  
3915 ["ltcc"] = 10918,  
3916 ["gtcc"] = 10919,  
3917 ["lescc"] = 10920,  
3918 ["gescc"] = 10921,  
3919 ["smt"] = 10922,  
3920 ["lat"] = 10923,  
3921 ["smtE"] = 10924,  
3922 ["late"] = 10925,  
3923 ["bumpE"] = 10926,  
3924 ["pre"] = 10927,  
3925 ["preceq"] = 10927,  
3926 ["PrecedesEqual"] = 10927,  
3927 ["sce"] = 10928,  
3928 ["succeq"] = 10928,  
3929 ["SucceedsEqual"] = 10928,  
3930 ["prE"] = 10931,  
3931 ["scE"] = 10932,  
3932 ["prnE"] = 10933,  
3933 ["precneqq"] = 10933,  
3934 ["scnE"] = 10934,  
3935 ["succneqq"] = 10934,  
3936 ["prap"] = 10935,  
3937 ["precapprox"] = 10935,  
3938 ["scap"] = 10936,  
3939 ["succapprox"] = 10936,  
3940 ["prnap"] = 10937,  
3941 ["precnapprox"] = 10937,  
3942 ["scnap"] = 10938,  
3943 ["succnapprox"] = 10938,  
3944 ["Pr"] = 10939,

3945 ["Sc"] = 10940,  
 3946 ["subdot"] = 10941,  
 3947 ["supdot"] = 10942,  
 3948 ["subplus"] = 10943,  
 3949 ["supplus"] = 10944,  
 3950 ["submult"] = 10945,  
 3951 ["supmult"] = 10946,  
 3952 ["subedot"] = 10947,  
 3953 ["supedot"] = 10948,  
 3954 ["subE"] = 10949,  
 3955 ["subseteqq"] = 10949,  
 3956 ["supE"] = 10950,  
 3957 ["supseteqq"] = 10950,  
 3958 ["subsim"] = 10951,  
 3959 ["supsim"] = 10952,  
 3960 ["subnE"] = 10955,  
 3961 ["subsetneqq"] = 10955,  
 3962 ["supnE"] = 10956,  
 3963 ["supsetneqq"] = 10956,  
 3964 ["csub"] = 10959,  
 3965 ["csup"] = 10960,  
 3966 ["csube"] = 10961,  
 3967 ["csupe"] = 10962,  
 3968 ["subsup"] = 10963,  
 3969 ["supsup"] = 10964,  
 3970 ["subsub"] = 10965,  
 3971 ["supsup"] = 10966,  
 3972 ["suphsub"] = 10967,  
 3973 ["supdsub"] = 10968,  
 3974 ["forkv"] = 10969,  
 3975 ["topfork"] = 10970,  
 3976 ["mlcp"] = 10971,  
 3977 ["Dashv"] = 10980,  
 3978 ["DoubleLeftTee"] = 10980,  
 3979 ["Vdashl"] = 10982,  
 3980 ["Barv"] = 10983,  
 3981 ["vBar"] = 10984,  
 3982 ["vBarv"] = 10985,  
 3983 ["Vbar"] = 10987,  
 3984 ["Not"] = 10988,  
 3985 ["bNot"] = 10989,  
 3986 ["rnmid"] = 10990,  
 3987 ["cirmid"] = 10991,  
 3988 ["midcir"] = 10992,  
 3989 ["topcir"] = 10993,  
 3990 ["nhpar"] = 10994,  
 3991 ["parsim"] = 10995,

3992 ["parsl"] = 11005,  
3993 ["fflig"] = 64256,  
3994 ["filig"] = 64257,  
3995 ["fllig"] = 64258,  
3996 ["ffilig"] = 64259,  
3997 ["ffllig"] = 64260,  
3998 ["Ascr"] = 119964,  
3999 ["Cscr"] = 119966,  
4000 ["Dscr"] = 119967,  
4001 ["Gscr"] = 119970,  
4002 ["Jscr"] = 119973,  
4003 ["Kscr"] = 119974,  
4004 ["Nscr"] = 119977,  
4005 ["Oscr"] = 119978,  
4006 ["Pscr"] = 119979,  
4007 ["Qscr"] = 119980,  
4008 ["Sscr"] = 119982,  
4009 ["Tscr"] = 119983,  
4010 ["Uscr"] = 119984,  
4011 ["Vscr"] = 119985,  
4012 ["Wscr"] = 119986,  
4013 ["Xscr"] = 119987,  
4014 ["Yscr"] = 119988,  
4015 ["Zscr"] = 119989,  
4016 ["ascr"] = 119990,  
4017 ["bscr"] = 119991,  
4018 ["cscr"] = 119992,  
4019 ["dscr"] = 119993,  
4020 ["fscr"] = 119995,  
4021 ["hscr"] = 119997,  
4022 ["iscr"] = 119998,  
4023 ["jscr"] = 119999,  
4024 ["kscr"] = 120000,  
4025 ["lscr"] = 120001,  
4026 ["mscr"] = 120002,  
4027 ["nscr"] = 120003,  
4028 ["pscr"] = 120005,  
4029 ["qscr"] = 120006,  
4030 ["rscr"] = 120007,  
4031 ["sscr"] = 120008,  
4032 ["tscr"] = 120009,  
4033 ["uscr"] = 120010,  
4034 ["vscr"] = 120011,  
4035 ["wscr"] = 120012,  
4036 ["xscr"] = 120013,  
4037 ["yscr"] = 120014,  
4038 ["zscr"] = 120015,

4039 ["Afr"] = 120068,  
4040 ["Bfr"] = 120069,  
4041 ["Dfr"] = 120071,  
4042 ["Efr"] = 120072,  
4043 ["Ffr"] = 120073,  
4044 ["Gfr"] = 120074,  
4045 ["Jfr"] = 120077,  
4046 ["Kfr"] = 120078,  
4047 ["Lfr"] = 120079,  
4048 ["Mfr"] = 120080,  
4049 ["Nfr"] = 120081,  
4050 ["Ofr"] = 120082,  
4051 ["Pfr"] = 120083,  
4052 ["Qfr"] = 120084,  
4053 ["Sfr"] = 120086,  
4054 ["Tfr"] = 120087,  
4055 ["Ufr"] = 120088,  
4056 ["Vfr"] = 120089,  
4057 ["Wfr"] = 120090,  
4058 ["Xfr"] = 120091,  
4059 ["Yfr"] = 120092,  
4060 ["afr"] = 120094,  
4061 ["bfr"] = 120095,  
4062 ["cfr"] = 120096,  
4063 ["dfr"] = 120097,  
4064 ["efr"] = 120098,  
4065 ["ffr"] = 120099,  
4066 ["gfr"] = 120100,  
4067 ["hfr"] = 120101,  
4068 ["ifr"] = 120102,  
4069 ["jfr"] = 120103,  
4070 ["kfr"] = 120104,  
4071 ["lfr"] = 120105,  
4072 ["mfr"] = 120106,  
4073 ["nfr"] = 120107,  
4074 ["ofr"] = 120108,  
4075 ["pfr"] = 120109,  
4076 ["qfr"] = 120110,  
4077 ["rfr"] = 120111,  
4078 ["sfr"] = 120112,  
4079 ["tfr"] = 120113,  
4080 ["ufr"] = 120114,  
4081 ["vfr"] = 120115,  
4082 ["wfr"] = 120116,  
4083 ["xfr"] = 120117,  
4084 ["yfr"] = 120118,  
4085 ["zfr"] = 120119,

```
4086 ["Aopf"] = 120120,  
4087 ["Bopf"] = 120121,  
4088 ["Dopf"] = 120123,  
4089 ["Eopf"] = 120124,  
4090 ["Fopf"] = 120125,  
4091 ["Gopf"] = 120126,  
4092 ["Iopf"] = 120128,  
4093 ["Jopf"] = 120129,  
4094 ["Kopf"] = 120130,  
4095 ["Lopf"] = 120131,  
4096 ["Mopf"] = 120132,  
4097 ["Oopf"] = 120134,  
4098 ["Sopf"] = 120138,  
4099 ["Topf"] = 120139,  
4100 ["Uopf"] = 120140,  
4101 ["Vopf"] = 120141,  
4102 ["Wopf"] = 120142,  
4103 ["Xopf"] = 120143,  
4104 ["Yopf"] = 120144,  
4105 ["aopf"] = 120146,  
4106 ["bopf"] = 120147,  
4107 ["copf"] = 120148,  
4108 ["dopf"] = 120149,  
4109 ["eopf"] = 120150,  
4110 ["fopf"] = 120151,  
4111 ["gopf"] = 120152,  
4112 ["hopf"] = 120153,  
4113 ["iopf"] = 120154,  
4114 ["jopf"] = 120155,  
4115 ["kopf"] = 120156,  
4116 ["lopf"] = 120157,  
4117 ["mopf"] = 120158,  
4118 ["nopf"] = 120159,  
4119 ["oopf"] = 120160,  
4120 ["popf"] = 120161,  
4121 ["qopf"] = 120162,  
4122 ["ropf"] = 120163,  
4123 ["sopf"] = 120164,  
4124 ["topf"] = 120165,  
4125 ["uopf"] = 120166,  
4126 ["vopf"] = 120167,  
4127 ["wopf"] = 120168,  
4128 ["xopf"] = 120169,  
4129 ["yopf"] = 120170,  
4130 ["zopf"] = 120171,  
4131 }
```

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

```
4132 function entities.dec_entity(s)
4133   return unicode.utf8.char(tonumber(s))
4134 end
```

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

```
4135 function entities.hex_entity(s)
4136   return unicode.utf8.char(tonumber("0x"..s))
4137 end
```

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

```
4138 function entities.char_entity(s)
4139   local n = character_entities[s]
4140   if n == nil then
4141     return "&" .. s .. ";"
4142   end
4143   return unicode.utf8.char(n)
4144 end
```

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

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

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

```
4145 M.writer = {}
```

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

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

```
4146 function M.writer.new(options)
4147   local self = {}
```



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

```
4148 self.cacheDir = options.cacheDir
```

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

```
4149 self.hybrid = options.hybrid
```

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

```
4150 local slice_specifiers = {}
4151 for specifier in options.slice:gmatch("[^%s]+") do
4152   table.insert(slice_specifiers, specifier)
4153 end
4154
4155 if #slice_specifiers == 2 then
4156   self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4157   local slice_begin_type = self.slice_begin:sub(1, 1)
4158   if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4159     self.slice_begin = "^" .. self.slice_begin
4160   end
4161   local slice_end_type = self.slice_end:sub(1, 1)
4162   if slice_end_type ~= "^" and slice_end_type ~= "$" then
4163     self.slice_end = "$" .. self.slice_end
4164   end
4165 elseif #slice_specifiers == 1 then
4166   self.slice_begin = "^" .. slice_specifiers[1]
4167   self.slice_end = "$" .. slice_specifiers[1]
4168 end
4169
4170 if self.slice_begin == "^" and self.slice_end ~= "^" then
4171   self.is_writing = true
4172 else
4173   self.is_writing = false
4174 end
```

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

```
4175 self.suffix = ".tex"
```

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

```
4176 self.space = " "
```

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

```
4177 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
4178 function self.plain(s)
4179   return s
```

```

4180 end

Define writer->paragraph as a function that will transform an input paragraph
s to the output format.
4181 function self.paragraph(s)
4182   if not self.is_writing then return "" end
4183   return s
4184 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.
4185 function self.pack(name)
4186   return [[\input ]] .. name .. [[\relax]]
4187 end

Define writer->interblocksep as the output format of a block element separator.
4188 function self.interblocksep()
4189   if not self.is_writing then return "" end
4190   return "\\markdownRendererInterblockSeparator\n{}"
4191 end

Define writer->linebreak as the output format of a forced line break.
4192 self.linebreak = "\\markdownRendererLineBreak\n{}"

Define writer->ellipsis as the output format of an ellipsis.
4193 self.ellipsis = "\\markdownRendererEllipsis{}"

Define writer->hrule as the output format of a horizontal rule.
4194 function self.hrule()
4195   if not self.is_writing then return "" end
4196   return "\\markdownRendererHorizontalRule{}"
4197 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.
4198 self.escaped_uri_chars = {
4199   [{""] = "\\markdownRendererLeftBrace{}",
4200   ["}"] = "\\markdownRendererRightBrace{}",
4201   [%"] = "\\markdownRendererPercentSign{}",
4202   ["\\" = "\\markdownRendererBackslash{}",
4203   }
4204 self.escaped_minimal_strings = {
4205   ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4206   ["☒"] = "\\markdownRendererTickedBox{}",
4207   ["☐"] = "\\markdownRendererHalfTickedBox{}",
4208   ["□"] = "\\markdownRendererUntickedBox{}",
4209   }

```

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

```

4210 self.escaped_chars = {
4211   [{"}] = "\\markdownRendererLeftBrace{}",
4212   [{"}] = "\\markdownRendererRightBrace{}",
4213   [{"%"}] = "\\markdownRendererPercentSign{}",
4214   [{"\\"}] = "\\markdownRendererBackslash{}",
4215   [{"#"}] = "\\markdownRendererHash{}",
4216   [{"$"}] = "\\markdownRendererDollarSign{}",
4217   [{"&"}] = "\\markdownRendererAmpersand{}",
4218   [{"_"}] = "\\markdownRendererUnderscore{}",
4219   [{"^"}] = "\\markdownRendererCircumflex{}",
4220   [{"~"}] = "\\markdownRendererTilde{}",
4221   [{"|"}] = "\\markdownRendererPipe{}",
4222 }

```

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

```

4223 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4224 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4225 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.

```

4226 if options.hybrid then
4227   self.string = self.escape_minimal
4228   self.uri = self.escape_minimal
4229 else
4230   self.string = self.escape
4231   self.uri = self.escape_uri
4232 end

```

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

```

4233 function self.code(s)
4234   return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
4235 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.

```

4236 function self.link(lab,src,tit)

```

```

4237     return {"\\markdownRendererLink{" ,lab,"} ",
4238             "{" ,self.escape(src),"} ",
4239             "{" ,self.uri(src),"} ",
4240             "{" ,self.string(tit or ""),"} "}
4241 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.

```

4242 function self.image(lab,src,tit)
4243     return {"\\markdownRendererImage{" ,lab,"} ",
4244             "{" ,self.string(src),"} ",
4245             "{" ,self.uri(src),"} ",
4246             "{" ,self.string(tit or ""),"} "}
4247 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.

```

4248 local function ulitem(s)
4249     return {"\\markdownRendererULItem " ,s,
4250             "\\markdownRendererULItemEnd "}
4251 end
4252
4253 function self.bulletlist(items,tight)
4254     if not self.is_writing then return "" end
4255     local buffer = {}
4256     for _,item in ipairs(items) do
4257         buffer[#buffer + 1] = ulitem(item)
4258     end
4259     local contents = util.intersperse(buffer,"\\n")
4260     if tight and options.tightLists then
4261         return {"\\markdownRendererULBeginTight\\n" ,contents,
4262                 "\\n\\markdownRendererULEndTight "}
4263     else
4264         return {"\\markdownRendererULBegin\\n" ,contents,
4265                 "\\n\\markdownRendererULEnd "}
4266     end
4267 end

```

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

```

4268 local function olitem(s,num)
4269     if num ~= nil then
4270         return {"\\markdownRendererOLItemWithNumber{" ,num,"} " ,s,

```

```

4271         "\\markdownRenderer01ItemEnd "}
4272     else
4273         return {"\\markdownRenderer01Item ",s,
4274             "\\markdownRenderer01ItemEnd "}
4275     end
4276 end
4277
4278 function self.orderedlist(items,tight,startnum)
4279     if not self.is_writing then return "" end
4280     local buffer = {}
4281     local num = startnum
4282     for _,item in ipairs(items) do
4283         buffer[#buffer + 1] = olitem(item,num)
4284         if num ~= nil then
4285             num = num + 1
4286         end
4287     end
4288     local contents = util.intersperse(buffer,"\n")
4289     if tight and options.tightLists then
4290         return {"\\markdownRenderer01BeginTight\n",contents,
4291             "\n\\markdownRenderer01EndTight "}
4292     else
4293         return {"\\markdownRenderer01Begin\n",contents,
4294             "\n\\markdownRenderer01End "}
4295     end
4296 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.

```

4297 function self.inline_html_comment(contents)
4298     return {"\\markdownRendererInlineHtmlComment{",contents,""}
4299 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.

```

4300 function self.block_html_comment(contents)
4301     if not self.is_writing then return "" end
4302     return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4303         "\n\\markdownRendererBlockHtmlCommentEnd "}
4304 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.

```

4305 function self.inline_html_tag(contents)
4306     return {"\\markdownRendererInlineHtmlTag{",self.string(contents),""}

```

```
4307 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.

```
4308 function self.block_html_element(s)
4309   if not self.is_writing then return "" end
4310   local name = util.cache(self.cacheDir, s, nil, nil, ".verbatim")
4311   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
4312 end
```

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

```
4313 function self.emphasis(s)
4314   return {"\\markdownRendererEmphasis{" ,s,"}"}
4315 end
```

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

```
4316 function self.checkbox(f)
4317   if f == 1.0 then
4318     return "☒ "
4319   elseif f == 0.0 then
4320     return "☐ "
4321   else
4322     return "◻ "
4323   end
4324 end
```

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

```
4325 function self.strong(s)
4326   return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4327 end
```

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

```
4328 function self.blockquote(s)
4329   if #util.ropetostring(s) == 0 then return "" end
4330   return {"\\markdownRendererBlockQuoteBegin\\n",s,
4331     "\\n\\markdownRendererBlockQuoteEnd "}
4332 end
```

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

```
4333 function self.verbatim(s)
4334   if not self.is_writing then return "" end
4335   s = string.gsub(s, '[\r\n%s]*$', '')
4336   local name = util.cache(self.cacheDir, s, nil, nil, ".verbatim")
```

```

4337     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4338 end

```

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

```

4339 function self.document(d)
4340     local active_attributes = self.active_attributes
4341     local buf = {"\\markdownRendererDocumentBegin\n", d}
4342
4343     -- pop attributes for sections that have ended
4344     if options.headerAttributes and self.is_writing then
4345         while #active_attributes > 0 do
4346             local attributes = active_attributes[#active_attributes]
4347             if #attributes > 0 then
4348                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4349             end
4350             table.remove(active_attributes, #active_attributes)
4351         end
4352     end
4353
4354     table.insert(buf, "\\markdownRendererDocumentEnd")
4355
4356     return buf
4357 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.

```

4358 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.

```

4359 function self.heading(s, level, attributes)
4360     attributes = attributes or {}
4361     for i = 1, #attributes do
4362         attributes[attributes[i]] = true
4363     end
4364
4365     local active_attributes = self.active_attributes
4366     local slice_begin_type = self.slice_begin:sub(1, 1)
4367     local slice_begin_identifier = self.slice_begin:sub(2) or ""
4368     local slice_end_type = self.slice_end:sub(1, 1)
4369     local slice_end_identifier = self.slice_end:sub(2) or ""
4370
4371     local buf = {}
4372
4373     -- push empty attributes for implied sections
4374     while #active_attributes < level-1 do
4375         table.insert(active_attributes, {})

```

```

4376     end
4377
4378     -- pop attributes for sections that have ended
4379     while #active_attributes >= level do
4380         local active_identifiers = active_attributes[#active_attributes]
4381         -- tear down all active attributes at slice end
4382         if active_identifiers["#" .. slice_end_identifier] ~= nil
4383             and slice_end_type == "$" then
4384             for header_level = #active_attributes, 1, -1 do
4385                 if options.headerAttributes and #active_attributes[header_level] > 0 then
4386                     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4387                 end
4388             end
4389             self.is_writing = false
4390         end
4391         table.remove(active_attributes, #active_attributes)
4392         if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4393             table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4394         end
4395         -- apply all active attributes at slice beginning
4396         if active_identifiers["#" .. slice_begin_identifier] ~= nil
4397             and slice_begin_type == "$" then
4398             for header_level = 1, #active_attributes do
4399                 if options.headerAttributes and #active_attributes[header_level] > 0 then
4400                     table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4401                 end
4402             end
4403             self.is_writing = true
4404         end
4405     end
4406
4407     -- tear down all active attributes at slice end
4408     if attributes["#" .. slice_end_identifier] ~= nil
4409         and slice_end_type == "^" then
4410         for header_level = #active_attributes, 1, -1 do
4411             if options.headerAttributes and #active_attributes[header_level] > 0 then
4412                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4413             end
4414         end
4415         self.is_writing = false
4416     end
4417
4418     -- push attributes for the new section
4419     table.insert(active_attributes, attributes)
4420     if self.is_writing and options.headerAttributes and #attributes > 0 then
4421         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4422     end

```



```

4423
4424 -- apply all active attributes at slice beginning
4425 if attributes["#" .. slice_begin_identifer] ~= nil
4426     and slice_begin_type == "^" then
4427     for header_level = 1, #active_attributes do
4428         if options.headerAttributes and #active_attributes[header_level] > 0 then
4429             table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4430         end
4431     end
4432     self.is_writing = true
4433 end
4434
4435 if self.is_writing then
4436     table.sort(attributes)
4437     local key, value
4438     for i = 1, #attributes do
4439         if attributes[i]:sub(1, 1) == "#" then
4440             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
4441                 attributes[i]:sub(2), "}"}))
4442         elseif attributes[i]:sub(1, 1) == "." then
4443             table.insert(buf, {"\\markdownRendererAttributeName{" ,
4444                 attributes[i]:sub(2), "}"}))
4445         else
4446             key, value = attributes[i]:match("(%w+)=(%w+)")
4447             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
4448                 key, "}{" , value, "}"}))
4449         end
4450     end
4451 end
4452
4453 local cmd
4454 level = level + options.shiftHeadings
4455 if level <= 1 then
4456     cmd = "\\markdownRendererHeadingOne"
4457 elseif level == 2 then
4458     cmd = "\\markdownRendererHeadingTwo"
4459 elseif level == 3 then
4460     cmd = "\\markdownRendererHeadingThree"
4461 elseif level == 4 then
4462     cmd = "\\markdownRendererHeadingFour"
4463 elseif level == 5 then
4464     cmd = "\\markdownRendererHeadingFive"
4465 elseif level >= 6 then
4466     cmd = "\\markdownRendererHeadingSix"
4467 else
4468     cmd = ""
4469 end

```

```

4470     if self.is_writing then
4471         table.insert(buf, {cmd, "{", s, "}"})
4472     end
4473
4474     return buf
4475 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.

```

4476 function self.get_state()
4477     return {
4478         is_writing=self.is_writing,
4479         active_attributes={table.unpack(self.active_attributes)},
4480     }
4481 end

```

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

```

4482 function self.set_state(s)
4483     local previous_state = self.get_state()
4484     for key, value in pairs(s) do
4485         self[key] = value
4486     end
4487     return previous_state
4488 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`.

```

4489 function self.defer_call(f)
4490     local previous_state = self.get_state()
4491     return function(...)
4492         local state = self.set_state(previous_state)
4493         local return_value = f(...)
4494         self.set_state(state)
4495         return return_value
4496     end
4497 end
4498
4499 return self
4500 end

```

### 3.1.4 Parsers

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

```

4501 local parsers = {}

```

### 3.1.4.1 Basic Parsers

```
4502 parsers.percent           = P("%")
4503 parsers.at                 = P("@")
4504 parsers.comma               = P(",")
4505 parsers.asterisk            = P("*")
4506 parsers.dash                = P("-")
4507 parsers.plus                = P("+")
4508 parsers.underscore          = P("_")
4509 parsers.period              = P(".")
4510 parsers.hash                 = P("#")
4511 parsers.ampersand           = P("&")
4512 parsers.backtick            = P("`")
4513 parsers.less                 = P("<")
4514 parsers.more                 = P(">")
4515 parsers.space                = P(" ")
4516 parsers.squote              = P("'")
4517 parsers.dquote              = P('"')
4518 parsers.lparent              = P("(")
4519 parsers.rparent              = P(")")
4520 parsers.lbracket             = P("[")
4521 parsers.rbracket             = P("]")
4522 parsers.lbrace                = P("{")
4523 parsers.rbrace                = P("}")
4524 parsers.circumflex           = P("^")
4525 parsers.slash                = P("/")
4526 parsers.equal                = P("=")
4527 parsers.colon                = P(":")
4528 parsers.semicolon            = P(";")
4529 parsers.exclamation          = P("!")
4530 parsers.pipe                  = P("|")
4531 parsers.tilde                 = P("~")
4532 parsers.backslash            = P("\\")
4533 parsers.tab                   = P("\t")
4534 parsers.newline               = P("\n")
4535 parsers.tightblocksep        = P("\001")
4536
4537 parsers.digit                 = R("09")
4538 parsers.hexdigit              = R("09", "af", "AF")
4539 parsers.letter                = R("AZ", "az")
4540 parsers.alphanumeric          = R("AZ", "az", "09")
4541 parsers.keyword                = parsers.letter
4542                               * parsers.alphanumeric^0
4543 parsers.internal_punctuation = S(";, .?")
4544
4545 parsers.doubleasterisks       = P("**")
4546 parsers.doubleunderscores     = P("__")
4547 parsers.fourspace             = P("    ")
```

```

4548
4549 parsers.any = P(1)
4550 parsers.fail = parsers.any - 1
4551
4552 parsers.escapable = S("\\`*_{}[]()+_!.<>#-~:~@;")
4553 parsers.anyescaped = parsers.backslash / "" * parsers.escapable
4554 + parsers.any
4555
4556 parsers.spacechar = S("\t ")
4557 parsers.spacing = S(" \n\r\t")
4558 parsers.nonspacechar = parsers.any - parsers.spacing
4559 parsers.optionalspace = parsers.spacechar^0
4560
4561 parsers.normalchar = parsers.any - (V("SpecialChar")
4562 + parsers.spacing
4563 + parsers.tightblocksep)
4564 parsers.eof = -parsers.any
4565 parsers.nonindentpace = parsers.space^-3 * - parsers.spacechar
4566 parsers.indent = parsers.space^-3 * parsers.tab
4567 + parsers.fourspace / ""
4568 parsers.linechar = P(1 - parsers.newline)
4569
4570 parsers.blankline = parsers.optionalspace
4571 * parsers.newline / "\n"
4572 parsers.blanklines = parsers.blankline^0
4573 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0
4574 parsers.indentedline = parsers.indent / ""
4575 * C(parsers.linechar^1 * parsers.newline^-
1)
4576 parsers.optionallyindentedline = parsers.indent^-1 / ""
4577 * C(parsers.linechar^1 * parsers.newline^-
1)
4578 parsers.sp = parsers.spacing^0
4579 parsers.spnl = parsers.optionalspace
4580 * (parsers.newline * parsers.optionalspace)^-
1
4581 parsers.line = parsers.linechar^0 * parsers.newline
4582 parsers.nonemptyline = parsers.line - parsers.blankline

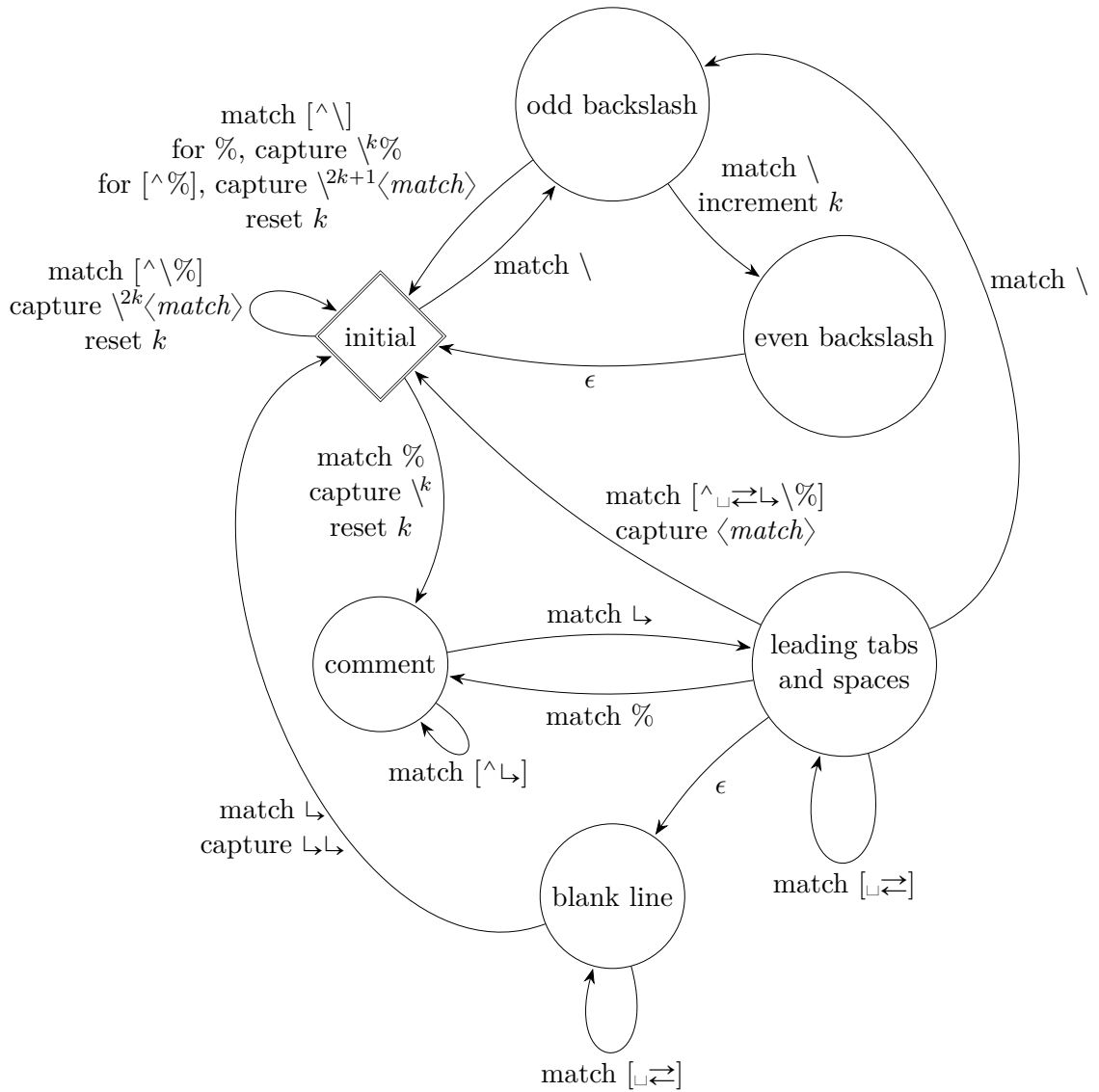
```

The `parsers.commented_line^1` parser recognizes the regular language of  $\text{T}_{\text{E}}\text{X}$  comments, see an equivalent finite automaton in Figure 6.

```

4583 parsers.commented_line_letter = parsers.linechar
4584 + parsers.newline
4585 - parsers.backslash
4586 - parsers.percent
4587 parsers.commented_line = Cg(Cc(""), "backslashes")
4588 * ((#(parsers.commented_line_letter

```



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

```

4589         - parsers.newline)
4590     * Cb("backslashes")
4591     * Cs(parsers.commented_line_letter
4592         - parsers.newline)^1 -- initial
4593     * Cg(Cc(""), "backslashes")
4594 + #(parsers.backslash * parsers.backslash)
4595 * Cg((parsers.backslash -- even backslash
4596     * parsers.backslash)^1, "backslashes")
4597 + (parsers.backslash
4598     * (#parsers.percent
4599     * Cb("backslashes")
4600     / function(backslashes)
4601     return string.rep("\\", #backslashes / 2)
4602     end
4603     * C(parsers.percent)
4604     + #parsers.commented_line_letter
4605     * Cb("backslashes")
4606     * Cc("\\")
4607     * C(parsers.commented_line_letter))
4608     * Cg(Cc(""), "backslashes"))^0
4609 * (#parsers.percent
4610 * Cb("backslashes")
4611 / function(backslashes)
4612     return string.rep("\\", #backslashes / 2)
4613     end
4614 * ((parsers.percent -- comment
4615     * parsers.line
4616     * #parsers.blankline) -- blank line
4617     / "\n"
4618     + parsers.percent -- comment
4619     * parsers.line
4620     * parsers.optionalspace) -- leading tabs and spaces
4621 + #(parsers.newline)
4622 * Cb("backslashes")
4623 * C(parsers.newline))
4624
4625 parsers.chunk = parsers.line * (parsers.optionallyindentedline
4626     - parsers.blankline)^0
4627
4628 parsers.css_identifier_char = R("AZ", "az", "09") + S("-_")
4629 parsers.css_identifier = (parsers.hash + parsers.period)
4630 * (((parsers.css_identifier_char
4631     - parsers.dash - parsers.digit)
4632     * parsers.css_identifier_char^1)
4633 + (parsers.dash
4634     * (parsers.css_identifier_char
4635     - parsers.digit)

```

```

4636                                     * parsers.css_identifier_char^0))
4637 parsers.attribute_key_char      = parsers.any - parsers.space
4638                                 - parsers.squote - parsers.dquote
4639                                 - parsers.more - parsers.slash
4640                                 - parsers.equal
4641 parsers.attribute_value_char    = parsers.any - parsers.space
4642                                 - parsers.dquote - parsers.more
4643
4644 -- block followed by 0 or more optionally
4645 -- indented blocks with first line indented.
4646 parsers.indented_blocks = function(bl)
4647   return Cs( bl
4648             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
4649             * (parsers.blankline^1 + parsers.eof) )
4650 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

4651 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
4652
4653 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
4654                  * (parsers.tab + parsers.space^-3)
4655                  + parsers.space * parsers.bulletchar * #parsers.spacing
4656                  * (parsers.tab + parsers.space^-2)
4657                  + parsers.space * parsers.space * parsers.bulletchar
4658                  * #parsers.spacing
4659                  * (parsers.tab + parsers.space^-1)
4660                  + parsers.space * parsers.space * parsers.space
4661                  * parsers.bulletchar * #parsers.spacing
4662                  )
4663
4664 local function tickbox(interior)
4665   return parsers.optionalspace * parsers.lbracket
4666         * interior * parsers.rbracket * parsers.spacechar^1
4667 end
4668
4669 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
4670 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
4671 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
4672

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

4673 parsers.openticks = Cg(parsers.backtick^1, "ticks")
4674
4675 local function captures_equal_length(_,i,a,b)
4676   return #a == #b and i
4677 end

```

```

4678
4679 parsers.closeticks = parsers.space^-1
4680                       * Cmt(C(parsers.backtick^1)
4681                           * Cb("ticks"), captures_equal_length)
4682
4683 parsers.intickschar = (parsers.any - S("\n\r`"))
4684                       + (parsers.newline * -parsers.blankline)
4685                       + (parsers.space - parsers.closeticks)
4686                       + (parsers.backtick^1 - parsers.closeticks)
4687
4688 parsers.inticks      = parsers.openticks * parsers.space^-1
4689                       * C(parsers.intickschar^0) * parsers.closeticks

```

### 3.1.4.4 Parsers Used for Fenced Code Blocks

```

4690 local function captures_geq_length(_,i,a,b)
4691   return #a >= #b and i
4692 end
4693
4694 parsers.infostring   = (parsers.linechar - (parsers.backtick
4695       + parsers.space^1 * (parsers.newline + parsers.eof)))^0
4696
4697 local fenceindent
4698 parsers.fencehead    = function(char)
4699   return             C(parsers.nonindentospace) / function(s) fenceindent = #s end
4700   * Cg(char^3, "fencelength")
4701   * parsers.optionalspace * C(parsers.infostring)
4702   * parsers.optionalspace * (parsers.newline + parsers.eof)
4703 end
4704
4705 parsers.fencetail    = function(char)
4706   return             parsers.nonindentospace
4707   * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
4708   * parsers.optionalspace * (parsers.newline + parsers.eof)
4709   + parsers.eof
4710 end
4711
4712 parsers.fencedline   = function(char)
4713   return             C(parsers.line - parsers.fencetail(char))
4714   / function(s)
4715     local i = 1
4716     local remaining = fenceindent
4717     while true do
4718       local c = s:sub(i, i)
4719       if c == " " and remaining > 0 then
4720         remaining = remaining - 1
4721         i = i + 1

```



```

4722         elsif c == "\t" and remaining > 3 then
4723             remaining = remaining - 4
4724             i = i + 1
4725         else
4726             break
4727         end
4728     end
4729     return s:sub(i)
4730 end
4731 end

```

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```

4732 parsers.leader      = parsers.space^-3
4733
4734 -- content in balanced brackets, parentheses, or quotes:
4735 parsers.bracketed  = P{ parsers.lbracket
4736                       * ((parsers.anyescaped - (parsers.lbracket
4737                                                         + parsers.rbracket
4738                                                         + parsers.blankline^2)
4739                           ) + V(1))^0
4740                       * parsers.rbracket }
4741
4742 parsers.inparens    = P{ parsers.lparent
4743                       * ((parsers.anyescaped - (parsers.lparent
4744                                                         + parsers.rparent
4745                                                         + parsers.blankline^2)
4746                           ) + V(1))^0
4747                       * parsers.rparent }
4748
4749 parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
4750                       * ((parsers.anyescaped - (parsers.squote
4751                                                         + parsers.blankline^2)
4752                           ) + V(1))^0
4753                       * parsers.squote }
4754
4755 parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
4756                       * ((parsers.anyescaped - (parsers.dquote
4757                                                         + parsers.blankline^2)
4758                           ) + V(1))^0
4759                       * parsers.dquote }
4760
4761 -- bracketed tag for markdown links, allowing nested brackets:
4762 parsers.tag         = parsers.lbracket
4763                       * Cs((parsers.alphanumeric^1
4764                             + parsers.bracketed
4765                             + parsers.inticks

```

```

4766             + (parsers.anyescaped
4767               - (parsers.rbracket + parsers.blankline^2))^0)
4768         * parsers.rbracket
4769
4770 -- url for markdown links, allowing nested brackets:
4771 parsers.url      = parsers.less * Cs((parsers.anyescaped
4772                                   - parsers.more)^0)
4773                                   * parsers.more
4774                                   + Cs((parsers.inparens + (parsers.anyescaped
4775                                       - parsers.spacing
4776                                       - parsers.rparent))^1)
4777
4778 -- quoted text, possibly with nested quotes:
4779 parsers.title_s  = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
4780                                       + parsers.squoted)^0)
4781                                       * parsers.squote
4782
4783 parsers.title_d  = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
4784                                       + parsers.dquoted)^0)
4785                                       * parsers.dquote
4786
4787 parsers.title_p  = parsers.lparent
4788                 * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
4789                 * parsers.rparent
4790
4791 parsers.title    = parsers.title_d + parsers.title_s + parsers.title_p
4792
4793 parsers.optionaltitle
4794                 = parsers.spnl * parsers.title * parsers.spacechar^0
4795                 + Cc("")

```

### 3.1.4.6 Parsers Used for HTML

```

4796 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
4797 parsers.keyword_exact = function(s)
4798   local parser = P(0)
4799   for i=1,#s do
4800     local c = s:sub(i,i)
4801     local m = c .. upper(c)
4802     parser = parser * S(m)
4803   end
4804   return parser
4805 end
4806
4807 parsers.block_keyword =
4808   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
4809   parsers.keyword_exact("center") + parsers.keyword_exact("del") +

```

```

4810 parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
4811 parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
4812 parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
4813 parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
4814 parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
4815 parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
4816 parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
4817 parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
4818 parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
4819 parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
4820 parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
4821 parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
4822 parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
4823 parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
4824 parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
4825 parsers.keyword_exact("td") + parsers.keyword_exact("tr")
4826
4827 -- There is no reason to support bad html, so we expect quoted attributes
4828 parsers.htmlattributevalue
4829         = parsers.squote * (parsers.any - (parsers.blankline
4830                                           + parsers.squote))^0
4831           * parsers.squote
4832         + parsers.dquote * (parsers.any - (parsers.blankline
4833                                           + parsers.dquote))^0
4834           * parsers.dquote
4835
4836 parsers.htmlattribute = parsers.spacing^1
4837                       * (parsers.alphanumeric + S("_-"))^1
4838                       * parsers.sp * parsers.equal * parsers.sp
4839                       * parsers.htmlattributevalue
4840
4841 parsers.htmlcomment  = P("<!--")
4842                       * parsers.optionalspace
4843                       * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
4844                       * parsers.optionalspace
4845                       * P("-->")
4846
4847 parsers.htmlinstruction = P("<?") * (parsers.any - P(">"))^0 * P(">")
4848
4849 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
4850                   * parsers.sp * parsers.more
4851
4852 parsers.openelt_exact = function(s)
4853   return parsers.less * parsers.sp * parsers.keyword_exact(s)
4854         * parsers.htmlattribute^0 * parsers.sp * parsers.more
4855 end
4856

```

```

4857 parsers.openelt_block = parsers.sp * parsers.block_keyword
4858             * parsers.htmlattribute^0 * parsers.sp * parsers.more
4859
4860 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
4861             * parsers.keyword * parsers.sp * parsers.more
4862
4863 parsers.closeelt_exact = function(s)
4864   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
4865         * parsers.sp * parsers.more
4866 end
4867
4868 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
4869             * parsers.htmlattribute^0 * parsers.sp * parsers.slash
4870             * parsers.more
4871
4872 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
4873             * parsers.htmlattribute^0 * parsers.sp * parsers.slash
4874             * parsers.more
4875
4876 parsers.displaytext = (parsers.any - parsers.less)^1
4877
4878 -- return content between two matched HTML tags
4879 parsers.in_matched = function(s)
4880   return { parsers.openelt_exact(s)
4881         * (V(1) + parsers.displaytext
4882         + (parsers.less - parsers.closeelt_exact(s)))^0
4883         * parsers.closeelt_exact(s) }
4884 end
4885
4886 local function parse_matched_tags(s,pos)
4887   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
4888   return lpeg.match(parsers.in_matched(t),s,pos-1)
4889 end
4890
4891 parsers.in_matched_block_tags = parsers.less
4892             * Cmt(#parsers.openelt_block, parse_matched_tags)
4893

```

### 3.1.4.7 Parsers Used for HTML Entities

```

4894 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
4895             * C(parsers.hexdigit^1) * parsers.semicolon
4896 parsers.decentity = parsers.ampersand * parsers.hash
4897             * C(parsers.digit^1) * parsers.semicolon
4898 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
4899             * parsers.semicolon

```

### 3.1.4.8 Helpers for References

```
4900 -- parse a reference definition: [foo]: /bar "title"
4901 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
4902                                 * parsers.spacechar^0 * parsers.url
4903                                 * parsers.optionaltitle * parsers.blankline^1
```

### 3.1.4.9 Inline Elements

```
4904 parsers.Inline          = V("Inline")
4905 parsers.IndentedInline = V("IndentedInline")
4906
4907 -- parse many p between starter and ender
4908 parsers.between = function(p, starter, ender)
4909   local ender2 = B(parsers.nonspacechar) * ender
4910   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
4911 end
4912
4913 parsers.urlchar          = parsers.anyescaped - parsers.newline - parsers.more
```

### 3.1.4.10 Block Elements

```
4914 parsers.lineof = function(c)
4915   return (parsers.leader * (P(c) * parsers.optionalspace)^3
4916         * (parsers.newline * parsers.blankline^1
4917         + parsers.newline^-1 * parsers.eof))
4918 end
```

### 3.1.4.11 Lists

#### 3.1.4.12 Headings

```
4919 parsers.heading_attribute = C(parsers.css_identifier)
4920                           + C((parsers.attribute_key_char
4921                               - parsers.rbrace)^1
4922                               * parsers.equal
4923                               * (parsers.attribute_value_char
4924                               - parsers.rbrace)^1)
4925 parsers.HeadingAttributes = parsers.lbrace
4926                           * parsers.heading_attribute
4927                           * (parsers.spacechar^1
4928                           * parsers.heading_attribute)^0
4929                           * parsers.rbrace
4930
4931 -- parse Atx heading start and return level
4932 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
4933                     * -parsers.hash / length
4934
```

```

4935 -- parse setext header ending and return level
4936 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
4937
4938 local function strip_atx_end(s)
4939   return s:gsub("#%s*\n$", "")
4940 end

```

### 3.1.5 Markdown Reader

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

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

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

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

```

4941 M.reader = {}
4942 function M.reader.new(writer, options, extensions)
4943   local self = {}

```

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

```

4944   self.writer = writer

```

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.

```

4945   self.parsers = {}
4946   (function(parsers)
4947     setmetatable(self.parsers, {
4948       __index = function (_, key)
4949         return parsers[key]
4950       end
4951     })
4952   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.

```

4953   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.

```
4954 function self.normalize_tag(tag)
4955     return string.lower(
4956         gsub(util.ropo_to_string(tag), "[ \\n\\r\\t]+", " "))
4957 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.

```
4958 local function iterlines(s, f)
4959     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
4960     return util.ropo_to_string(rope)
4961 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.

```
4962 if options.preserveTabs then
4963     self.expandtabs = function(s) return s end
4964 else
4965     self.expandtabs = function(s)
4966         if s:find("\t") then
4967             return iterlines(s, util.expand_tabs_in_line)
4968         else
4969             return s
4970         end
4971     end
4972 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.

```
4973 self.parser_functions = {}
4974 self.create_parser = function(name, grammar, toplevel)
4975     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.

```
4976         if toplevel and options.stripIndent then
4977             local min_prefix_length, min_prefix = nil, ''
```

```

4978         str = iterlines(str, function(line)
4979             if lpeg.match(parsers.nonemptyline, line) == nil then
4980                 return line
4981             end
4982             line = util.expand_tabs_in_line(line)
4983             local prefix = lpeg.match(C(parsers.optionalspace), line)
4984             local prefix_length = #prefix
4985             local is_shorter = min_prefix_length == nil
4986             is_shorter = is_shorter or prefix_length < min_prefix_length
4987             if is_shorter then
4988                 min_prefix_length, min_prefix = prefix_length, prefix
4989             end
4990             return line
4991         end)
4992         str = str:gsub('^' .. min_prefix, '')
4993     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.

```

4994         if toplevel and (options.texComments or options.hybrid) then
4995             str = lpeg.match(Ct(parsers.commented_line^1), str)
4996             str = util.rope_to_string(str)
4997         end
4998         local res = lpeg.match(grammar(), str)
4999         if res == nil then
5000             error(format("%s failed on:\n%s", name, str:sub(1,20)))
5001         else
5002             return res
5003         end
5004     end
5005 end
5006
5007 self.create_parser("parse_blocks",
5008     function()
5009         return parsers.blocks
5010     end, true)
5011
5012 self.create_parser("parse_blocks_nested",
5013     function()
5014         return parsers.blocks_nested
5015     end, false)
5016
5017 self.create_parser("parse_inlines",
5018     function()
5019         return parsers.inlines
5020     end, false)

```



```

5021
5022 self.create_parser("parse_inlines_no_link",
5023                     function()
5024                         return parsers.inlines_no_link
5025                     end, false)
5026
5027 self.create_parser("parse_inlines_no_inline_note",
5028                     function()
5029                         return parsers.inlines_no_inline_note
5030                     end, false)
5031
5032 self.create_parser("parse_inlines_no_html",
5033                     function()
5034                         return parsers.inlines_no_html
5035                     end, false)
5036
5037 self.create_parser("parse_inlines_nbsp",
5038                     function()
5039                         return parsers.inlines_nbsp
5040                     end, false)

```

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

```

5041 if options.hashEnumerators then
5042     parsers.dig = parsers.digit + parsers.hash
5043 else
5044     parsers.dig = parsers.digit
5045 end
5046
5047 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5048 + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5049     * (parsers.tab + parsers.space^1)
5050 + C(parsers.dig * parsers.period) * #parsers.spacing
5051     * (parsers.tab + parsers.space^-2)
5052 + parsers.space * C(parsers.dig^2 * parsers.period)
5053     * #parsers.spacing
5054 + parsers.space * C(parsers.dig * parsers.period)
5055     * #parsers.spacing
5056     * (parsers.tab + parsers.space^-1)
5057 + parsers.space * parsers.space * C(parsers.dig^1
5058     * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

5059 -- strip off leading > and indents, and run through blocks
5060 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
5061 1)/""
5061     * parsers.linechar^0 * parsers.newline)^1

```

```

5062             * (-(parsers.leader * parsers.more
5063               + parsers.blankline) * parsers.linechar^1
5064               * parsers.newline)^0
5065
5066     if not options.breakableBlockquotes then
5067       parsers.blockquote_body = parsers.blockquote_body
5068       * (parsers.blankline^0 / "")
5069     end

```

### 3.1.5.5 Parsers Used for Footnotes (local)

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

```

5070 -- List of references defined in the document
5071 local references
5072
5073 -- add a reference to the list
5074 local function register_link(tag,url,title)
5075     references[self.normalize_tag(tag)] = { url = url, title = title }
5076     return ""
5077 end
5078
5079 -- lookup link reference and return either
5080 -- the link or nil and fallback text.
5081 local function lookup_reference(label,sps,tag)
5082     local tagpart
5083     if not tag then
5084         tag = label
5085         tagpart = ""
5086     elseif tag == "" then
5087         tag = label
5088         tagpart = "[]"
5089     else
5090         tagpart = {"[",
5091                 self.parser_functions.parse_inlines(tag),
5092                 "]" }
5093     end
5094     if sps then
5095         tagpart = {sps, tagpart}
5096     end
5097     local r = references[self.normalize_tag(tag)]
5098     if r then
5099         return r
5100     else
5101         return nil, {"[",
5102                 self.parser_functions.parse_inlines(label),
5103                 "]", tagpart}

```

```

5104     end
5105 end
5106
5107 -- lookup link reference and return a link, if the reference is found,
5108 -- or a bracketed label otherwise.
5109 local function indirect_link(label,sps,tag)
5110     return writer.defer_call(function()
5111         local r,fallback = lookup_reference(label,sps,tag)
5112         if r then
5113             return writer.link(
5114                 self.parser_functions.parse_inlines_no_link(label),
5115                 r.url, r.title)
5116         else
5117             return fallback
5118         end
5119     end)
5120 end
5121
5122 -- lookup image reference and return an image, if the reference is found,
5123 -- or a bracketed label otherwise.
5124 local function indirect_image(label,sps,tag)
5125     return writer.defer_call(function()
5126         local r,fallback = lookup_reference(label,sps,tag)
5127         if r then
5128             return writer.image(writer.string(label), r.url, r.title)
5129         else
5130             return {"!", fallback}
5131         end
5132     end)
5133 end

```

### 3.1.5.7 Inline Elements (local)

```

5134 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5135                  / writer.string
5136
5137 parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
5138                  / writer.string
5139
5140 parsers.Ellipsis = P("...") / writer.ellipsis
5141
5142 parsers.Smart    = parsers.Ellipsis
5143
5144 parsers.Code     = parsers.inticks / writer.code
5145
5146 if options.blankBeforeBlockquote then
5147     parsers.bqstart = parsers.fail

```

```

5148 else
5149     parsers.bqstart = parsers.more
5150 end
5151
5152 if options.blankBeforeHeading then
5153     parsers.headerstart = parsers.fail
5154 else
5155     parsers.headerstart = parsers.hash
5156                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5157                         * parsers.optionalspace * parsers.newline)
5158 end
5159
5160 parsers.EndlineExceptions
5161     = parsers.blankline -- paragraph break
5162     + parsers.tightblocksep -- nested list
5163     + parsers.eof      -- end of document
5164     + parsers.bqstart
5165     + parsers.headerstart
5166
5167 parsers.Endline = parsers.newline
5168                 * -V("EndlineExceptions")
5169                 * parsers.spacechar^0
5170                 / (options.hardLineBreaks and writer.linebreak
5171                   or writer.space)
5172
5173 parsers.OptionalIndent
5174     = parsers.spacechar^1 / writer.space
5175
5176 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5177               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5178               + parsers.spacechar^1 * parsers.Endline
5179               * parsers.optionalspace
5180               / (options.hardLineBreaks
5181                 and writer.linebreak
5182                 or writer.space)
5183               + parsers.spacechar^1 * parsers.optionalspace
5184               / writer.space
5185
5186 parsers.NonbreakingEndline
5187     = parsers.newline
5188     * -V("EndlineExceptions")
5189     * parsers.spacechar^0
5190     / (options.hardLineBreaks and writer.linebreak
5191       or writer.nbsp)
5192
5193 parsers.NonbreakingSpace
5194     = parsers.spacechar^2 * parsers.Endline / writer.linebreak

```

```

5195         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5196         + parsers.spacechar^1 * parsers.Endline
5197             * parsers.optionalspace
5198             / (options.hardLineBreaks
5199               and writer.linebreak
5200               or writer.nbsp)
5201         + parsers.spacechar^1 * parsers.optionalspace
5202             / writer.nbsp
5203
5204     if options.underscores then
5205         parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5206                                           parsers.doubleasterisks)
5207                           + parsers.between(parsers.Inline, parsers.doubleunderscores,
5208                                           parsers.doubleunderscores)
5209                           ) / writer.strong
5210
5211         parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5212                                           parsers.asterisk)
5213                           + parsers.between(parsers.Inline, parsers.underscore,
5214                                           parsers.underscore)
5215                           ) / writer.emphasis
5216     else
5217         parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5218                                           parsers.doubleasterisks)
5219                           ) / writer.strong
5220
5221         parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5222                                           parsers.asterisk)
5223                           ) / writer.emphasis
5224     end
5225
5226     parsers.AutoLinkUrl = parsers.less
5227                         * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5228                         * parsers.more
5229                         / function(url)
5230                             return writer.link(writer.escape(url), url)
5231                         end
5232
5233     parsers.AutoLinkEmail = parsers.less
5234                          * C((parsers.alphanumeric + S("-._+"))^1
5235                              * P("@") * parsers.urlchar^1)
5236                          * parsers.more
5237                          / function(email)
5238                              return writer.link(writer.escape(email),
5239                                                  "mailto:".email)
5240                          end
5241

```

```

5242 parsers.AutoLinkRelativeReference
5243     = parsers.less
5244     * C(parsers.urlchar^1)
5245     * parsers.more
5246     / function(url)
5247         return writer.link(writer.escape(url), url)
5248     end
5249
5250 parsers.DirectLink    = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5251     * parsers.spnl
5252     * parsers.lparent
5253     * (parsers.url + Cc("")) -- link can be empty [foo]()
5254     * parsers.optionaltitle
5255     * parsers.rparent
5256     / writer.link
5257
5258 parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
1
5259     / indirect_link
5260
5261 -- parse a link or image (direct or indirect)
5262 parsers.Link          = parsers.DirectLink + parsers.IndirectLink
5263
5264 parsers.DirectImage  = parsers.exclamation
5265     * (parsers.tag / self.parser_functions.parse_inlines)
5266     * parsers.spnl
5267     * parsers.lparent
5268     * (parsers.url + Cc("")) -- link can be empty [foo]()
5269     * parsers.optionaltitle
5270     * parsers.rparent
5271     / writer.image
5272
5273 parsers.IndirectImage = parsers.exclamation * parsers.tag
5274     * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5275
5276 parsers.Image         = parsers.DirectImage + parsers.IndirectImage
5277
5278 -- avoid parsing long strings of * or _ as emph/strong
5279 parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
5280     / writer.string
5281
5282 parsers.EscapedChar  = parsers.backslash * C(parsers.escapable) / writer.string
5283
5284 parsers.InlineHtml   = parsers.emptyelt_any / writer.inline_html_tag
5285     + (parsers.htmlcomment / self.parser_functions.parse_inlines_no.
5286     / writer.inline_html_comment
5287     + parsers.htmlinstruction

```

```

5288         + parsers.openelt_any / writer.inline_html_tag
5289         + parsers.closeelt_any / writer.inline_html_tag
5290
5291     parsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
5292         + parsers.decentity / entities.dec_entity / writer.string
5293         + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.8 Block Elements (local)

```

5294     parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_nest
5295         / writer.block_html_comment
5296         + parsers.emptyelt_block / writer.block_html_element
5297         + parsers.openelt_exact("hr") / writer.block_html_element
5298         + parsers.in_matched_block_tags / writer.block_html_element
5299         + parsers.htmlinstruction
5300
5301     parsers.Verbatim = Cs( (parsers.blanklines
5302         * ((parsers.indentedline - parsers.blankline))^1)^1
5303         ) / self.expandtabs / writer.verbatim
5304
5305     parsers.Blockquote = Cs(parsers.blockquote_body^1)
5306         / self.parser_functions.parse_blocks_nested
5307         / writer.blockquote
5308
5309     parsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
5310         + parsers.lineof(parsers.dash)
5311         + parsers.lineof(parsers.underscore)
5312         ) / writer.hrule
5313
5314     parsers.Reference = parsers.define_reference_parser / register_link
5315
5316     parsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline^1)
5317         * ( parsers.newline
5318         * ( parsers.blankline^1
5319         + #parsers.hash
5320         + #(parsers.leader * parsers.more * parsers.space^-
5321         1)
5321         + parsers.eof
5322         )
5323         + parsers.eof )
5324         / writer.paragraph
5325
5326     parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
5327         / writer.plain

```

### 3.1.5.9 Lists (local)

```

5328     parsers.starter = parsers.bullet + parsers.enumerator

```

```

5329
5330 if options.taskLists then
5331     parsers.tickbox = ( parsers.ticked_box
5332                       + parsers.halfticked_box
5333                       + parsers.unticked_box
5334                       ) / writer.tickbox
5335 else
5336     parsers.tickbox = parsers.fail
5337 end
5338
5339 -- we use \001 as a separator between a tight list item and a
5340 -- nested list under it.
5341 parsers.NestedList      = Cs((parsers.optionallyindentedline
5342                             - parsers.starter)^1)
5343                       / function(a) return "\001"..a end
5344
5345 parsers.ListBlockLine  = parsers.optionallyindentedline
5346                       - parsers.blankline - (parsers.indent^-1
5347                                               * parsers.starter)
5348
5349 parsers.ListBlock      = parsers.line * parsers.ListBlockLine^0
5350
5351 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5352                                       * parsers.ListBlock
5353
5354 parsers.TightListItem = function(starter)
5355     return -parsers.HorizontalRule
5356           * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Nested
5357               1)
5358              / self.parser_functions.parse_blocks_nested)
5359           * -(parsers.blanklines * parsers.indent)
5360 end
5361
5362 parsers.LooseListItem = function(starter)
5363     return -parsers.HorizontalRule
5364           * Cs( starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
5365               * (parsers.NestedList + parsers.ListContinuationBlock^0)
5366               * (parsers.blanklines / "\n\n")
5367               ) / self.parser_functions.parse_blocks_nested
5368 end
5369
5370 parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
5371                       * parsers.skipblanklines * -parsers.bullet
5372                       + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5373                       * parsers.skipblanklines )
5374 / writer.bulletlist

```



```

5375 local function ordered_list(items,tight,startNumber)
5376   if options.startNumber then
5377     startNumber = tonumber(startNumber) or 1 -- fallback for '#'
5378     if startNumber ~= nil then
5379       startNumber = math.floor(startNumber)
5380     end
5381   else
5382     startNumber = nil
5383   end
5384   return writer.orderedlist(items,tight,startNumber)
5385 end
5386
5387 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
5388   ( Ct(parsers.TightListItem(Cb("listtype")))
5389     * parsers.TightListItem(parsers.enumerator)^0)
5390   * Cc(true) * parsers.skipblanklines * -parsers.enumerator
5391   + Ct(parsers.LooseListItem(Cb("listtype")))
5392     * parsers.LooseListItem(parsers.enumerator)^0)
5393   * Cc(false) * parsers.skipblanklines
5394   ) * Cb("listtype") / ordered_list

```

### 3.1.5.10 Blank (local)

```

5395 parsers.Blank = parsers.blankline / ""
5396               + parsers.Reference
5397               + (parsers.tightblocksep / "\n")

```

### 3.1.5.11 Headings (local)

```

5398 -- parse atx header
5399 parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
5400                   * parsers.optionalspace
5401                   * (C(parsers.line)
5402                     / strip_atx_end
5403                     / self.parser_functions.parse_inlines)
5404                   * Cb("level")
5405                   / writer.heading
5406
5407 parsers.SetextHeading = #(parsers.line * S("-"))
5408                       * Ct(parsers.linechar^1
5409                           / self.parser_functions.parse_inlines)
5410                       * parsers.newline
5411                       * parsers.HeadingLevel
5412                       * parsers.optionalspace
5413                       * parsers.newline
5414                       / writer.heading
5415
5416 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

**3.1.5.12 Syntax Specification** Create a `reader->syntax` hash table that stores the PEG grammar.

```
5417 self.syntax =
5418     { "Blocks",
5419
5420       Blocks = ( V("ExpectedJekyllData")
5421                 * (V("Blank")^0 / writer.interblocksep)
5422                 )^-1
5423                 * V("Blank")^0
5424                 * V("Block")^-1
5425                 * (V("Blank")^0 / writer.interblocksep
5426                   * V("Block"))^0
5427                 * V("Blank")^0 * parsers.eof,
5428
5429       Blank = parsers.Blank,
5430
5431       UnexpectedJekyllData = parsers.fail,
5432       ExpectedJekyllData = parsers.fail,
5433
5434       Block = V("ContentBlock")
5435              + V("UnexpectedJekyllData")
5436              + V("Blockquote")
5437              + V("PipeTable")
5438              + V("Verbatim")
5439              + V("FencedCode")
5440              + V("HorizontalRule")
5441              + V("BulletList")
5442              + V("OrderedList")
5443              + V("Heading")
5444              + V("DefinitionList")
5445              + V("DisplayHtml")
5446              + V("Paragraph")
5447              + V("Plain"),
5448
5449       ContentBlock = parsers.fail,
5450       Blockquote = parsers.Blockquote,
5451       Verbatim = parsers.Verbatim,
5452       FencedCode = parsers.fail,
5453       HorizontalRule = parsers.HorizontalRule,
5454       BulletList = parsers.BulletList,
5455       OrderedList = parsers.OrderedList,
5456       Heading = parsers.Heading,
5457       DefinitionList = parsers.fail,
5458       DisplayHtml = parsers.DisplayHtml,
5459       Paragraph = parsers.Paragraph,
5460       PipeTable = parsers.fail,
5461       Plain = parsers.Plain,
```

```

5462     EndlineExceptions = parsers.EndlineExceptions,
5463
5464     Inline = V("Str")
5465           + V("Space")
5466           + V("Endline")
5467           + V("U1OrStarLine")
5468           + V("Strong")
5469           + V("Emph")
5470           + V("InlineNote")
5471           + V("NoteRef")
5472           + V("Citations")
5473           + V("Link")
5474           + V("Image")
5475           + V("Code")
5476           + V("AutoLinkUrl")
5477           + V("AutoLinkEmail")
5478           + V("AutoLinkRelativeReference")
5479           + V("InlineHtml")
5480           + V("HtmlEntity")
5481           + V("EscapedChar")
5482           + V("Smart")
5483           + V("Symbol"),
5484
5485     IndentedInline = V("Str")
5486                   + V("OptionalIndent")
5487                   + V("Endline")
5488                   + V("U1OrStarLine")
5489                   + V("Strong")
5490                   + V("Emph")
5491                   + V("InlineNote")
5492                   + V("NoteRef")
5493                   + V("Citations")
5494                   + V("Link")
5495                   + V("Image")
5496                   + V("Code")
5497                   + V("AutoLinkUrl")
5498                   + V("AutoLinkEmail")
5499                   + V("AutoLinkRelativeReference")
5500                   + V("InlineHtml")
5501                   + V("HtmlEntity")
5502                   + V("EscapedChar")
5503                   + V("Smart")
5504                   + V("Symbol"),
5505
5506     Str = parsers.Str,
5507     Space = parsers.Space,
5508     OptionalIndent = parsers.OptionalIndent,

```

```

5509     Endline           = parsers.Endline,
5510     U1OrStarLine      = parsers.U1OrStarLine,
5511     Strong            = parsers.Strong,
5512     Emph              = parsers.Emph,
5513     InlineNote        = parsers.fail,
5514     NoteRef           = parsers.fail,
5515     Citations         = parsers.fail,
5516     Link              = parsers.Link,
5517     Image             = parsers.Image,
5518     Code              = parsers.Code,
5519     AutoLinkUrl       = parsers.AutoLinkUrl,
5520     AutoLinkEmail     = parsers.AutoLinkEmail,
5521     AutoLinkRelativeReference
5522     = parsers.AutoLinkRelativeReference,
5523     InlineHtml        = parsers.InlineHtml,
5524     HtmlEntity        = parsers.HtmlEntity,
5525     EscapedChar       = parsers.EscapedChar,
5526     Smart             = parsers.Smart,
5527     Symbol            = parsers.Symbol,
5528     SpecialChar       = parsers.fail,
5529 }

```

Define a hash table of all characters with a special meaning that can be escaped and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar.

```

5530 local special_characters = {"*", "~", "[", "]", "<", "!", "\\"}
5531 self.add_special_character = function(c)
5532     table.insert(special_characters, c)
5533     self.syntax.SpecialChar = S(table.concat(special_characters, ""))
5534 end
5535 self.syntax.SpecialChar = S(table.concat(special_characters, ""))

```

Apply syntax extensions.

```

5536 for _, extension in ipairs(extensions) do
5537     extension.extend_writer(writer)
5538     extension.extend_reader(self)
5539 end
5540
5541 if options.underscores then
5542     self.add_special_character("_")
5543 end
5544
5545 if not options.codeSpans then
5546     self.syntax.Code = parsers.fail
5547 end
5548
5549 if not options.html then
5550     self.syntax.DisplayHtml = parsers.fail

```

```

5551     self.syntax.InlineHtml = parsers.fail
5552     self.syntax.HtmlEntity = parsers.fail
5553 else
5554     self.add_special_character("&")
5555 end
5556
5557 if options.preserveTabs then
5558     options.stripIndent = false
5559 end
5560
5561 if not options.smartEllipses then
5562     self.syntax.Smart = parsers.fail
5563 else
5564     self.add_special_character(".")
5565 end
5566
5567 if not options.relativeReferences then
5568     self.syntax.AutoLinkRelativeReference = parsers.fail
5569 end
5570
5571 local blocks_nested_t = util.table_copy(self.syntax)
5572 blocks_nested_t.ExpectedJekyllData = parsers.fail
5573 parsers.blocks_nested = Ct(blocks_nested_t)
5574
5575 parsers.blocks = Ct(self.syntax)
5576
5577 local inlines_t = util.table_copy(self.syntax)
5578 inlines_t[1] = "Inlines"
5579 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
5580 parsers.inlines = Ct(inlines_t)
5581
5582 local inlines_no_link_t = util.table_copy(inlines_t)
5583 inlines_no_link_t.Link = parsers.fail
5584 parsers.inlines_no_link = Ct(inlines_no_link_t)
5585
5586 local inlines_no_inline_note_t = util.table_copy(inlines_t)
5587 inlines_no_inline_note_t.InlineNote = parsers.fail
5588 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
5589
5590 local inlines_no_html_t = util.table_copy(inlines_t)
5591 inlines_no_html_t.DisplayHtml = parsers.fail
5592 inlines_no_html_t.InlineHtml = parsers.fail
5593 inlines_no_html_t.HtmlEntity = parsers.fail
5594 parsers.inlines_no_html = Ct(inlines_no_html_t)
5595
5596 local inlines_nbsp_t = util.table_copy(inlines_t)
5597 inlines_nbsp_t.Endline = parsers.NonbreakingEndline

```

```

5598 inlines_nbsp_t.Space = parsers.NonbreakingSpace
5599 parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

**3.1.5.13 Exported Conversion Function** Define `reader->convert` as a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

5600 function self.convert(input)
5601     references = {}

```

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

```

5602     local opt_string = {}
5603     for k,_ in pairs(defaultOptions) do
5604         local v = options[k]
5605         if k ~= "cacheDir" then
5606             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5607         end
5608     end
5609     table.sort(opt_string)
5610     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
5611     local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain T<sub>E</sub>X output via the `writer->pack` method.

```

5612     local function convert(input)
5613         local document = self.parser_functions.parse_blocks(input)
5614         return util.rope_to_string(writer.document(document))
5615     end
5616     if options.eagerCache or options.finalizeCache then
5617         local name = util.cache(options.cacheDir, input, salt, convert, ".md" .. writer.s
5618         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

5619     else
5620         output = convert(input)
5621     end

```

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

```

5622     if options.finalizeCache then
5623         local file, mode
5624         if options.frozenCacheCounter > 0 then
5625             mode = "a"
5626         else
5627             mode = "w"

```

```

5628     end
5629     file = assert(io.open(options.frozenCacheFileName, mode),
5630         [[could not open file "]] .. options.frozenCacheFileName
5631         .. [{" for writing}])
5632     assert(file:write([[expandafter\global\expandafter\def\csname ]]
5633         .. [markdownFrozenCache]] .. options.frozenCacheCounter
5634         .. [\endcsname{]] .. output .. [{}]] .. "\n"))
5635     assert(file:close())
5636 end
5637 return output
5638 end
5639 return self
5640 end

```

### 3.1.6 Syntax Extensions for Markdown

Create `extensions` hash table that contains 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.

```

5641 M.extensions = {}

```

**3.1.6.1 Citations** The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is `true`, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

5642 M.extensions.citations = function(citation_nbsps)

```

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

```

5643 local escaped_citation_chars = {
5644     [{"}] = "\\markdownRendererLeftBrace{]",
5645     [{"}] = "\\markdownRendererRightBrace{]",
5646     [{"%}] = "\\markdownRendererPercentSign{]",
5647     [{"\\}] = "\\markdownRendererBackslash{]",
5648     [{"#}] = "\\markdownRendererHash{]",
5649 }
5650 return {
5651     extend_writer = function(self)

```

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

```

5652         local escape_citation = util.escaper(
5653             escaped_citation_chars,
5654             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.

```
5655     if self.hybrid then
5656         self.citation = self.escape_minimal
5657     else
5658         self.citation = escape_citation
5659     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.

```
5660     function self.citations(text_cites, cites)
5661         local buffer = {"\markdownRenderer", text_cites and "TextCite" or "Cite",
5662             "{", #cites, "}"}
5663         for _,cite in ipairs(cites) do
5664             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
5665                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
5666         end
5667         return buffer
5668     end
5669 end, extend_reader = function(self)
5670     local parsers = self.parsers
5671     local syntax = self.syntax
5672     local writer = self.writer
5673
5674     local citation_chars
5675         = parsers.alphanumeric
5676         + S("#$%&-+<>~/_")
5677
5678     local citation_name
5679         = Cs(parsers.dash^-1) * parsers.at
5680         * Cs(citation_chars
5681             * (((citation_chars + parsers.internal_punctuation
```



```

5682         - parsers.comma - parsers.semicolon)
5683     * -#((parsers.internal_punctuation - parsers.comma
5684         - parsers.semicolon)^0
5685         * -(citation_chars + parsers.internal_punctuation
5686         - parsers.comma - parsers.semicolon))^0
5687     * citation_chars)^-1)
5688
5689 local citation_body_prenote
5690     = Cs((parsers.alphanumeric^1
5691         + parsers.bracketed
5692         + parsers.inticks
5693         + (parsers.anyescaped
5694         - (parsers.rbracket + parsers.blankline^2))
5695         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
5696
5697 local citation_body_postnote
5698     = Cs((parsers.alphanumeric^1
5699         + parsers.bracketed
5700         + parsers.inticks
5701         + (parsers.anyescaped
5702         - (parsers.rbracket + parsers.semicolon
5703         + parsers.blankline^2))
5704         - (parsers.spnl * parsers.rbracket))^0)
5705
5706 local citation_body_chunk
5707     = citation_body_prenote
5708     * parsers.spnl * citation_name
5709     * (parsers.internal_punctuation - parsers.semicolon)^-
5710     1
5711     * parsers.spnl * citation_body_postnote
5712
5712 local citation_body
5713     = citation_body_chunk
5714     * (parsers.semicolon * parsers.spnl
5715     * citation_body_chunk)^0
5716
5717 local citation_headless_body_postnote
5718     = Cs((parsers.alphanumeric^1
5719         + parsers.bracketed
5720         + parsers.inticks
5721         + (parsers.anyescaped
5722         - (parsers.rbracket + parsers.at
5723         + parsers.semicolon + parsers.blankline^2))
5724         - (parsers.spnl * parsers.rbracket))^0)
5725
5726 local citation_headless_body
5727     = citation_headless_body_postnote

```

```

5728         * (parsers.sp * parsers.semicolon * parsers.spnl
5729         * citation_body_chunk)^0
5730
5731     local citations
5732         = function(text_cites, raw_cites)
5733         local function normalize(str)
5734             if str == "" then
5735                 str = nil
5736             else
5737                 str = (citation_nbsps and
5738                     self.parser_functions.parse_inlines_nbsp or
5739                     self.parser_functions.parse_inlines)(str)
5740             end
5741             return str
5742         end
5743
5744         local cites = {}
5745         for i = 1,#raw_cites,4 do
5746             cites[#cites+1] = {
5747                 prenote = normalize(raw_cites[i]),
5748                 suppress_author = raw_cites[i+1] == "-",
5749                 name = writer.citation(raw_cites[i+2]),
5750                 postnote = normalize(raw_cites[i+3]),
5751             }
5752         end
5753         return writer.citations(text_cites, cites)
5754     end
5755
5756     local TextCitations
5757         = Ct((parsers.spnl
5758         * Cc("")
5759         * citation_name
5760         * ((parsers.spnl
5761             * parsers.lbracket
5762             * citation_headless_body
5763             * parsers.rbracket) + Cc("")))^1)
5764         / function(raw_cites)
5765             return citations(true, raw_cites)
5766         end
5767
5768     local ParenthesizedCitations
5769         = Ct((parsers.spnl
5770         * parsers.lbracket
5771         * citation_body
5772         * parsers.rbracket)^1)
5773         / function(raw_cites)
5774             return citations(false, raw_cites)

```

```

5775             end
5776
5777     local Citations = TextCitations + ParenthesizedCitations
5778
5779     syntax.Citations = Citations
5780
5781     self.add_special_character("@")
5782     self.add_special_character("-")
5783 end
5784 }
5785 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.

```

5786 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.

```

5787     local languages_json = (function()
5788         local ran_ok, kpse = pcall(require, "kpse")
5789         if ran_ok then
5790             kpse.set_program_name("luatex")

```

If the KPathSea library is unavailable, perhaps because we are using LuaMetaTeX, we will only locate the `options.contentBlocksLanguageMap` in the current working directory:

```

5791     else
5792         kpse = {lookup=function(filename, _) return filename end}
5793     end
5794     local base, prev, curr
5795     for _, filename in ipairs{kpse.lookup(language_map, { all=true })} do
5796         local file = io.open(filename, "r")
5797         if not file then goto continue end
5798         local json = file:read("*all"):gsub('^[\n]-', '[%1]=')
5799         curr = (function()
5800             local _ENV={ json=json, load=load } -- luacheck: ignore _ENV
5801             return load("return "..json)()
5802         end)()
5803         if type(curr) == "table" then
5804             if base == nil then
5805                 base = curr
5806             else

```

```

5807         setmetatable(prev, { __index = curr })
5808     end
5809     prev = curr
5810 end
5811 ::continue::
5812 end
5813 return base or {}
5814 end()
5815
5816 return {
5817     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.

```

5818     function self.contentblock(src,suf,type,tit)
5819         if not self.is_writing then return "" end
5820         src = src..".."..suf
5821         suf = suf:lower()
5822         if type == "onlineimage" then
5823             return {"\markdownRendererContentBlockOnlineImage{"..suf.."},"",
5824                 {"",self.string(src),"},",
5825                 {"",self.uri(src),"},",
5826                 {"",self.string(tit or ""),"},"}
5827         elseif languages_json[suf] then
5828             return {"\markdownRendererContentBlockCode{"..suf.."},"",
5829                 {"",self.string(languages_json[suf]),"},",
5830                 {"",self.string(src),"},",
5831                 {"",self.uri(src),"},",
5832                 {"",self.string(tit or ""),"},"}
5833         else
5834             return {"\markdownRendererContentBlock{"..suf.."},"",
5835                 {"",self.string(src),"},",
5836                 {"",self.uri(src),"},",
5837                 {"",self.string(tit or ""),"},"}
5838         end
5839     end
5840 end, extend_reader = function(self)
5841     local parsers = self.parsers
5842     local syntax = self.syntax
5843     local writer = self.writer
5844
5845     local contentblock_tail
5846         = parsers.optionaltitle
5847         * (parsers.newline + parsers.eof)
5848

```

```

5849 -- case insensitive online image suffix:
5850 local onlineimagesuffix
5851     = (function(...)
5852         local parser = nil
5853         for _, suffix in ipairs({...}) do
5854             local pattern=nil
5855             for i=1,#suffix do
5856                 local char=suffix:sub(i,i)
5857                 char = S(char:lower()..char:upper())
5858                 if pattern == nil then
5859                     pattern = char
5860                 else
5861                     pattern = pattern * char
5862                 end
5863             end
5864             if parser == nil then
5865                 parser = pattern
5866             else
5867                 parser = parser + pattern
5868             end
5869         end
5870         return parser
5871     end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
5872
5873 -- online image url for iA Writer content blocks with mandatory suffix,
5874 -- allowing nested brackets:
5875 local onlineimageurl
5876     = (parsers.less
5877         * Cs((parsers.anyescaped
5878             - parsers.more
5879             - #(parsers.period
5880                 * onlineimagesuffix
5881                 * parsers.more
5882                 * contentblock_tail))^0)
5883         * parsers.period
5884         * Cs(onlineimagesuffix)
5885         * parsers.more
5886         + (Cs((parsers.inparens
5887             + (parsers.anyescaped
5888                 - parsers.spacing
5889                 - parsers.rparent
5890                 - #(parsers.period
5891                     * onlineimagesuffix
5892                     * contentblock_tail))))^0)
5893         * parsers.period
5894         * Cs(onlineimagesuffix))
5895     ) * Cc("onlineimage")

```

```

5896
5897 -- filename for iA Writer content blocks with mandatory suffix:
5898 local localfilepath
5899     = parsers.slash
5900     * Cs((parsers.anyescaped
5901         - parsers.tab
5902         - parsers.newline
5903         - #(parsers.period
5904             * parsers.alphanumeric^1
5905             * contentblock_tail))^1)
5906     * parsers.period
5907     * Cs(parsers.alphanumeric^1)
5908     * Cc("localfile")
5909
5910 local ContentBlock
5911     = parsers.leader
5912     * (localfilepath + onlineimageurl)
5913     * contentblock_tail
5914     / writer.contentblock
5915
5916 syntax.ContentBlock = ContentBlock
5917 end
5918 }
5919 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.

```

5920 M.extensions.definition_lists = function(tight_lists)
5921   return {
5922     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.

```

5923     local function dlitem(term, defs)
5924       local retVal = {"\markdownRendererDlItem{" ,term,""}
5925       for _, def in ipairs(defs) do
5926         retVal[#retVal+1] = {"\markdownRendererDlDefinitionBegin " ,def,
5927                               "\markdownRendererDlDefinitionEnd "}
5928       end
5929       retVal[#retVal+1] = "\markdownRendererDlItemEnd "
5930       return retVal
5931     end
5932

```

```

5933 function self.definitionlist(items,tight)
5934     if not self.is_writing then return "" end
5935     local buffer = {}
5936     for _,item in ipairs(items) do
5937         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
5938     end
5939     if tight and tight_lists then
5940         return {"\\markdownRendererDlBeginTight\n", buffer,
5941             "\n\\markdownRendererDlEndTight"}
5942     else
5943         return {"\\markdownRendererDlBegin\n", buffer,
5944             "\n\\markdownRendererDlEnd"}
5945     end
5946 end
5947 end, extend_reader = function(self)
5948     local parsers = self.parsers
5949     local syntax = self.syntax
5950     local writer = self.writer
5951
5952     local defstartchar = S("~:")
5953
5954     local defstart = ( defstartchar * #parsers.spacing
5955         * (parsers.tab + parsers.space^-3)
5956         + parsers.space * defstartchar * #parsers.spacing
5957         * (parsers.tab + parsers.space^-2)
5958         + parsers.space * parsers.space * defstartchar
5959         * #parsers.spacing
5960         * (parsers.tab + parsers.space^-1)
5961         + parsers.space * parsers.space * parsers.space
5962         * defstartchar * #parsers.spacing
5963     )
5964
5965     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
5966
5967     local function definition_list_item(term, defs, _)
5968         return { term = self.parser_functions.parse_inlines(term),
5969             definitions = defs }
5970     end
5971
5972     local DefinitionListItemLoose
5973         = C(parsers.line) * parsers.skipblanklines
5974         * Ct((defstart
5975             * parsers.indented_blocks(dlchunk)
5976             / self.parser_functions.parse_blocks_nested)^1)
5977         * Cc(false) / definition_list_item
5978
5979     local DefinitionListItemTight

```

```

5980         = C(parsers.line)
5981         * Ct((defstart * dlchunk
5982             / self.parser_functions.parse_blocks_nested)^1)
5983         * Cc(true) / definition_list_item
5984
5985     local DefinitionList
5986         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
5987           + Ct(DefinitionListItemTight^1)
5988           * (parsers.skipblanklines
5989             * -DefinitionListItemLoose * Cc(true))
5990           ) / writer.definitionlist
5991
5992     syntax.DefinitionList = DefinitionList
5993 end
5994 }
5995 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.

```

5996 M.extensions.fenced_code = function(blank_before_code_fence)
5997   return {
5998     extend_writer = function(self)

```

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

```

5999     function self.fencedCode(i, s)
6000       if not self.is_writing then return "" end
6001       s = string.gsub(s, '[\r\n%s]*$', '')
6002       local name = util.cache(self.cacheDir, s, nil, nil, ".verbatim")
6003       return {"\\markdownRendererInputFencedCode{" ,name,"}{" ,i,"}"}
6004     end
6005   end, extend_reader = function(self)
6006     local parsers = self.parsers
6007     local syntax = self.syntax
6008     local writer = self.writer
6009
6010     local function captures_geq_length(_,i,a,b)
6011       return #a >= #b and i
6012     end
6013
6014     local infostring = (parsers.linechar - (parsers.backtick
6015       + parsers.space^1 * (parsers.newline + parsers.eof)))^0
6016
6017     local fenceindent

```



```

6018     local fencehead      = function(char)
6019         return           C(parsers.nonindentSPACE) / function(s) fenceindent = #s end
6020         * Cg(char^3, "fencelength")
6021         * parsers.optionalspace * C(Infostring)
6022         * parsers.optionalspace * (parsers.newline + parsers.eof)
6023     end
6024
6025     local fencetail      = function(char)
6026         return           parsers.nonindentSPACE
6027         * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6028         * parsers.optionalspace * (parsers.newline + parsers.eof)
6029         + parsers.eof
6030     end
6031
6032     local fencedline    = function(char)
6033         return           C(parsers.line - fencetail(char))
6034         / function(s)
6035             local i = 1
6036             local remaining = fenceindent
6037             while true do
6038                 local c = s:sub(i, i)
6039                 if c == " " and remaining > 0 then
6040                     remaining = remaining - 1
6041                     i = i + 1
6042                 elseif c == "\t" and remaining > 3 then
6043                     remaining = remaining - 4
6044                     i = i + 1
6045                 else
6046                     break
6047                 end
6048             end
6049             return s:sub(i)
6050         end
6051     end
6052
6053     local TildeFencedCode
6054         = fencehead(parsers.tilde)
6055         * Cs(fencedline(parsers.tilde)^0)
6056         * fencetail(parsers.tilde)
6057
6058     local BacktickFencedCode
6059         = fencehead(parsers.backtick)
6060         * Cs(fencedline(parsers.backtick)^0)
6061         * fencetail(parsers.backtick)
6062
6063     local FencedCode = (TildeFencedCode
6064         + BacktickFencedCode)

```

```

6065             / function(infostring, code)
6066                 return writer.fencedCode(writer.string(infostring),
6067                                         self.expandtabs(code))
6068             end
6069
6070     syntax.FencedCode = FencedCode
6071
6072     local fencestart
6073     if blank_before_code_fence then
6074         fencestart = parsers.fail
6075     else
6076         fencestart = fencehead(parsers.backtick)
6077                     + fencehead(parsers.tilde)
6078     end
6079
6080     parsers.EndlineExceptions = parsers.EndlineExceptions + fencestart
6081     syntax.EndlineExceptions = parsers.EndlineExceptions
6082 end
6083 }
6084 end

```

**3.1.6.5 Footnotes** The `extensions.footnotes` function implements the Pandoc footnote and inline footnote syntax extensions. When the `footnote` parameter is `true`, the Pandoc footnote syntax extension will be enabled. When the `inline_footnotes` parameter is `true`, the Pandoc inline footnote syntax extension will be enabled.

```

6085 M.extensions.footnotes = function(footnotes, inline_footnotes)
6086     assert(footnotes or inline_footnotes)
6087     return {
6088         extend_writer = function(self)

```

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

```

6089         function self.note(s)
6090             return {"\\markdownRendererFootnote{",s,""}
6091         end
6092     end, extend_reader = function(self)
6093         local parsers = self.parsers
6094         local syntax = self.syntax
6095         local writer = self.writer
6096
6097         if footnotes then
6098             local function strip_first_char(s)
6099                 return s:sub(2)
6100             end
6101

```

```

6102     local RawNoteRef
6103         = #(parsers.lbracket * parsers.circumflex)
6104         * parsers.tag / strip_first_char
6105
6106     local rawnotes = {}
6107
6108     -- like indirect_link
6109     local function lookup_note(ref)
6110         return writer.defer_call(function()
6111             local found = rawnotes[self.normalize_tag(ref)]
6112             if found then
6113                 return writer.note(
6114                     self.parser_functions.parse_blocks_nested(found))
6115             else
6116                 return {"[",
6117                     self.parser_functions.parse_inlines("^" .. ref), "]}
6118             end
6119         end)
6120     end
6121
6122     local function register_note(ref,rawnote)
6123         rawnotes[self.normalize_tag(ref)] = rawnote
6124         return ""
6125     end
6126
6127     local NoteRef = RawNoteRef / lookup_note
6128
6129     local NoteBlock
6130         = parsers.leader * RawNoteRef * parsers.colon
6131         * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6132         / register_note
6133
6134     parsers.Blank = NoteBlock + parsers.Blank
6135     syntax.Blank = parsers.Blank
6136
6137     syntax.NoteRef = NoteRef
6138 end
6139 if inline_footnotes then
6140     local InlineNote
6141         = parsers.circumflex
6142         * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
6143         / writer.note
6144     syntax.InlineNote = InlineNote
6145 end
6146
6147 self.add_special_character("^")
6148 end

```

```

6149 }
6150 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.

```

6151 M.extensions.header_attributes = function()
6152   return {
6153     extend_writer = function()
6154     end, extend_reader = function(self)
6155       local parsers = self.parsers
6156       local syntax = self.syntax
6157       local writer = self.writer
6158
6159       parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
6160         * parsers.optionalspace
6161         * (C(((parsers.linechar
6162           - ((parsers.hash^1
6163             * parsers.optionalspace
6164             * parsers.HeadingAttributes^-1
6165             + parsers.HeadingAttributes)
6166             * parsers.optionalspace
6167             * parsers.newline)))
6168           * (parsers.linechar
6169             - parsers.hash
6170             - parsers.lbrace)^0)^1)
6171         / self.parser_functions.parse_inlines)
6172       * Cg(Ct(parsers.newline
6173         + (parsers.hash^1
6174           * parsers.optionalspace
6175           * parsers.HeadingAttributes^-1
6176           + parsers.HeadingAttributes)
6177           * parsers.optionalspace
6178           * parsers.newline), "attributes")
6179       * Cb("level")
6180       * Cb("attributes")
6181       / writer.heading
6182
6183       parsers.SetextHeading = #(parsers.line * S("--"))
6184         * (C(((parsers.linechar
6185           - (parsers.HeadingAttributes
6186             * parsers.optionalspace
6187             * parsers.newline)))
6188           * (parsers.linechar
6189             - parsers.lbrace)^0)^1)
6190         / self.parser_functions.parse_inlines)
6191       * Cg(Ct(parsers.newline

```

```

6192             + (parsers.HeadingAttributes
6193                * parsers.optionalspace
6194                * parsers.newline)), "attributes")
6195         * parsers.HeadingLevel
6196         * Cb("attributes")
6197         * parsers.optionalspace
6198         * parsers.newline
6199         / writer.heading
6200
6201     parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
6202     syntax.Heading = parsers.Heading
6203 end
6204 }
6205 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` is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata

```

6206 M.extensions.jekyll_data = function(expect_jekyll_data)
6207   return {
6208     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`.

```

6209     function self.jekyllData(d, t, p)
6210       if not self.is_writing then return "" end
6211
6212       local buf = {}
6213
6214       local keys = {}
6215       for k, _ in pairs(d) do
6216         table.insert(keys, k)
6217       end
6218       table.sort(keys)
6219
6220       if not p then
6221         table.insert(buf, "\\markdownRendererJekyllDataBegin")
6222       end
6223
6224       if #d > 0 then
6225         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
6226         table.insert(buf, self.uri(p or "null"))

```

```

6227         table.insert(buf, "}{")
6228         table.insert(buf, #keys)
6229         table.insert(buf, "}")
6230     else
6231         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
6232         table.insert(buf, self.uri(p or "null"))
6233         table.insert(buf, "}{")
6234         table.insert(buf, #keys)
6235         table.insert(buf, "}")
6236     end
6237
6238     for _, k in ipairs(keys) do
6239         local v = d[k]
6240         local typ = type(v)
6241         k = tostring(k or "null")
6242         if typ == "table" and next(v) ~= nil then
6243             table.insert(
6244                 buf,
6245                 self.jekyllData(v, t, k)
6246             )
6247         else
6248             k = self.uri(k)
6249             v = tostring(v)
6250             if typ == "boolean" then
6251                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6252                 table.insert(buf, k)
6253                 table.insert(buf, "}{")
6254                 table.insert(buf, v)
6255                 table.insert(buf, "}")
6256             elseif typ == "number" then
6257                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6258                 table.insert(buf, k)
6259                 table.insert(buf, "}{")
6260                 table.insert(buf, v)
6261                 table.insert(buf, "}")
6262             elseif typ == "string" then
6263                 table.insert(buf, "\\markdownRendererJekyllDataString{")
6264                 table.insert(buf, k)
6265                 table.insert(buf, "}{")
6266                 table.insert(buf, t(v))
6267                 table.insert(buf, "}")
6268             elseif typ == "table" then
6269                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
6270                 table.insert(buf, k)
6271                 table.insert(buf, "}")
6272             else
6273                 error(format("Unexpected type %s for value of " ..

```

```

6274             "YAML key %s", typ, k))
6275         end
6276     end
6277 end
6278
6279     if #d > 0 then
6280         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6281     else
6282         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6283     end
6284
6285     if not p then
6286         table.insert(buf, "\\markdownRendererJekyllDataEnd")
6287     end
6288
6289     return buf
6290 end
6291 end, extend_reader = function(self)
6292     local parsers = self.parsers
6293     local syntax = self.syntax
6294     local writer = self.writer
6295
6296     local JekyllData
6297         = Cmt( C((parsers.line - P("----") - P("..."))^0)
6298             , function(s, i, text) -- luacheck: ignore s i
6299                 local data
6300                 local ran_ok, _ = pcall(function()
6301                     local tinyyaml = require("markdown-tinyyaml")
6302                     data = tinyyaml.parse(text, {timestamps=false})
6303                 end)
6304                 if ran_ok and data ~= nil then
6305                     return true, writer.jekyllData(data, function(s)
6306                         return self.parser_functions.parse_blocks_nested(s)
6307                     end, nil)
6308                 else
6309                     return false
6310                 end
6311             end
6312         )
6313
6314     local UnexpectedJekyllData
6315         = P("----")
6316         * parsers.blankline / 0
6317         * #(-parsers.blankline) -- if followed by blank, it's an hrule
6318         * JekyllData
6319         * (P("----") + P("..."))
6320

```

```

6321     local ExpectedJekyllData
6322         = ( P("---")
6323             * parsers.blankline / 0
6324             * #(-parsers.blankline) -- if followed by blank, it's an hrule
6325             )^-1
6326             * JekyllData
6327             * (P("---") + P("..."))^-1
6328
6329     syntax.UnexpectedJekyllData = UnexpectedJekyllData
6330     if expect_jekyll_data then
6331         syntax.ExpectedJekyllData = ExpectedJekyllData
6332     end
6333 end
6334 }
6335 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 parameter `table_captions` is `true`, the function also implements the Pandoc `table_captions` syntax extension for table captions.

```

6336 M.extensions.pipe_tables = function(table_captions)
6337
6338     local function make_pipe_table_rectangular(rows)
6339         local num_columns = #rows[2]
6340         local rectangular_rows = {}
6341         for i = 1, #rows do
6342             local row = rows[i]
6343             local rectangular_row = {}
6344             for j = 1, num_columns do
6345                 rectangular_row[j] = row[j] or ""
6346             end
6347             table.insert(rectangular_rows, rectangular_row)
6348         end
6349         return rectangular_rows
6350     end
6351
6352     local function pipe_table_row(allow_empty_first_column
6353                                   , nonempty_column
6354                                   , column_separator
6355                                   , column)
6356         local row_beginning
6357         if allow_empty_first_column then
6358             row_beginning = -- empty first column
6359                             #(parsers.spacechar^4
6360                               * column_separator)
6361         * parsers.optionalspace

```



```

6362         * column
6363         * parsers.optionalspace
6364         -- non-empty first column
6365         + parsers.nonindentspace
6366         * nonempty_column^-1
6367         * parsers.optionalspace
6368     else
6369         row_beginning = parsers.nonindentspace
6370         * nonempty_column^-1
6371         * parsers.optionalspace
6372     end
6373
6374     return Ct(row_beginning
6375         * (-- single column with no leading pipes
6376           #(column_separator
6377             * parsers.optionalspace
6378             * parsers.newline)
6379           * column_separator
6380           * parsers.optionalspace
6381           -- single column with leading pipes or
6382           -- more than a single column
6383           + (column_separator
6384             * parsers.optionalspace
6385             * column
6386             * parsers.optionalspace)^1
6387           * (column_separator
6388             * parsers.optionalspace)^-1))
6389     end
6390
6391     return {
6392         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.

```

6393         function self.table(rows, caption)
6394             if not self.is_writing then return "" end
6395             local buffer = {"\markdownRendererTable{",
6396                 caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}" }
6397             local temp = rows[2] -- put alignments on the first row
6398             rows[2] = rows[1]
6399             rows[1] = temp
6400             for i, row in ipairs(rows) do
6401                 table.insert(buffer, "{")
6402                 for _, column in ipairs(row) do
6403                     if i > 1 then -- do not use braces for alignments
6404                         table.insert(buffer, "{")

```

```

6405         end
6406         table.insert(buffer, column)
6407         if i > 1 then
6408             table.insert(buffer, "}")
6409         end
6410     end
6411     table.insert(buffer, "}")
6412 end
6413 return buffer
6414 end
6415 end, extend_reader = function(self)
6416     local parsers = self.parsers
6417     local syntax = self.syntax
6418     local writer = self.writer
6419
6420     local table_hline_separator = parsers.pipe + parsers.plus
6421
6422     local table_hline_column = (parsers.dash
6423         - #(parsers.dash
6424             * (parsers.spacechar
6425                 + table_hline_separator
6426                 + parsers.newline)))^1
6427     * (parsers.colon * Cc("r")
6428         + parsers.dash * Cc("d"))
6429     + parsers.colon
6430     * (parsers.dash
6431         - #(parsers.dash
6432             * (parsers.spacechar
6433                 + table_hline_separator
6434                 + parsers.newline)))^1
6435     * (parsers.colon * Cc("c")
6436         + parsers.dash * Cc("l"))
6437
6438     local table_hline = pipe_table_row(false
6439         , table_hline_column
6440         , table_hline_separator
6441         , table_hline_column)
6442
6443     local table_caption_beginning = parsers.skipblanklines
6444         * parsers.nonindentSPACE
6445         * (P("Table")^-1 * parsers.colon)
6446         * parsers.optionalspace
6447
6448     local table_row = pipe_table_row(true
6449         , (C((parsers.linechar - parsers.pipe)^1)
6450             / self.parser_functions.parse_inlines)
6451         , parsers.pipe

```

```

6452         , (C((parsers.linechar - parsers.pipe)^0)
6453           / self.parser_functions.parse_inlines))
6454
6455     local table_caption
6456     if table_captions then
6457         table_caption = #table_caption_beginning
6458                       * table_caption_beginning
6459                       * Ct(parsers.IndentedInline^1)
6460                       * parsers.newline
6461     else
6462         table_caption = parsers.fail
6463     end
6464
6465     local PipeTable = Ct(table_row * parsers.newline
6466                       * table_hline
6467                       * (parsers.newline * table_row)^0)
6468                       / make_pipe_table_rectangular
6469                       * table_caption^-1
6470                       / writer.table
6471
6472     syntax.PipeTable = PipeTable
6473 end
6474 }
6475 end

```

### 3.1.7 Conversion from Markdown to Plain T<sub>E</sub>X

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

```

6476 function M.new(options)

```

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

```

6477     options = options or {}
6478     setmetatable(options, { __index = function (_, key)
6479         return defaultOptions[key] end })
6480 % \par
6481 % \begin{markdown}
6482 %
6483 % Apply syntax extensions based on `options`.
6484 %
6485 % \end{markdown}
6486 % \begin{macrocode}
6487     local extensions = {}
6488
6489     if options.citations then
6490         local citations_extension = M.extensions.citations(options.citationNbsps)

```

```

6491     table.insert(extensions, citations_extension)
6492 end
6493
6494 if options.contentBlocks then
6495     local content_blocks_extension = M.extensions.content_blocks(
6496         options.contentBlocksLanguageMap)
6497     table.insert(extensions, content_blocks_extension)
6498 end
6499
6500 if options.definitionLists then
6501     local definition_lists_extension = M.extensions.definition_lists(
6502         options.tightLists)
6503     table.insert(extensions, definition_lists_extension)
6504 end
6505
6506 if options.fencedCode then
6507     local fenced_code_extension = M.extensions.fenced_code(
6508         options.blankBeforeCodeFence)
6509     table.insert(extensions, fenced_code_extension)
6510 end
6511
6512 if options.footnotes or options.inlineFootnotes then
6513     local footnotes_extension = M.extensions.footnotes(
6514         options.footnotes, options.inlineFootnotes)
6515     table.insert(extensions, footnotes_extension)
6516 end
6517
6518 if options.headerAttributes then
6519     local header_attributes_extension = M.extensions.header_attributes()
6520     table.insert(extensions, header_attributes_extension)
6521 end
6522
6523 if options.jekyllData then
6524     local jekyll_data_extension = M.extensions.jekyll_data(
6525         options.expectJekyllData)
6526     table.insert(extensions, jekyll_data_extension)
6527 end
6528
6529 if options.pipeTables then
6530     local pipe_tables_extension = M.extensions.pipe_tables(
6531         options.tableCaptions)
6532     table.insert(extensions, pipe_tables_extension)
6533 end
6534
6535 local writer = M.writer.new(options)
6536 local reader = M.reader.new(writer, options, extensions)
6537

```

```

6538 return reader.convert
6539 end
6540
6541 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 2.1.5.

```

6542
6543 local input
6544 if input_filename then
6545   local input_file = assert(io.open(input_filename, "r"),
6546     [[could not open file ]] .. input_filename .. [[ for reading]])
6547   input = assert(input_file:read("*a"))
6548   assert(input_file:close())
6549 else
6550   input = assert(io.read("*a"))
6551 end
6552

```

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

```

6553 local lfs = require("lfs")
6554 if options.cacheDir and not lfs.isdir(options.cacheDir) then
6555   assert(lfs.mkdir(options["cacheDir"]))
6556 end
6557
6558 local ran_ok, kpse = pcall(require, "kpse")
6559 if ran_ok then kpse.set_program_name("luatex") end
6560 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.

```

6561 if metadata.version ~= md.metadata.version then
6562   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
6563     "markdown.lua " .. md.metadata.version .. ".")
6564 end
6565 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.

```

6566 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
6567
6568 if output_filename then
6569   local output_file = assert(io.open(output_filename, "w"),
6570     [[could not open file ]] .. output_filename .. [[ for writing]])
6571   assert(output_file:write(output))
6572   assert(output_file:close())

```

```

6573 else
6574   assert(io.write(output))
6575 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

```

6576 \ifx\markdownInfo\undefined
6577   \def\markdownInfo#1{%
6578     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
6579 \fi
6580 \ifx\markdownWarning\undefined
6581   \def\markdownWarning#1{%
6582     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
6583 \fi
6584 \ifx\markdownError\undefined
6585   \def\markdownError#1#2{%
6586     \errhelp{#2.}}%
6587     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
6588 \fi

```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

6589 \def\markdownRendererInterblockSeparatorPrototype{\par}%
6590 \def\markdownRendererLineBreakPrototype{\hfil\break}%
6591 \let\markdownRendererEllipsisPrototype\dots
6592 \def\markdownRendererNbspPrototype{~}%
6593 \def\markdownRendererLeftBracePrototype{\char`{}%
6594 \def\markdownRendererRightBracePrototype{\char`}%
6595 \def\markdownRendererDollarSignPrototype{\char`$}%
6596 \def\markdownRendererPercentSignPrototype{\char`\}%
6597 \def\markdownRendererAmpersandPrototype{\&}%
6598 \def\markdownRendererUnderscorePrototype{\char`_%
6599 \def\markdownRendererHashPrototype{\char`#}%
6600 \def\markdownRendererCircumflexPrototype{\char`^}%
6601 \def\markdownRendererBackslashPrototype{\char`\}%
6602 \def\markdownRendererTildePrototype{\char`~}%
6603 \def\markdownRendererPipePrototype{|}%
6604 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
6605 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%

```

```

6606 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
6607   \markdownInput{#3}}%
6608 \def\markdownRendererContentBlockOnlineImagePrototype{%
6609   \markdownRendererImage}%
6610 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
6611   \markdownRendererInputFencedCode{#3}{#2}}%
6612 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
6613 \def\markdownRendererUlBeginPrototype{}%
6614 \def\markdownRendererUlBeginTightPrototype{}%
6615 \def\markdownRendererUlItemPrototype{}%
6616 \def\markdownRendererUlItemEndPrototype{}%
6617 \def\markdownRendererUlEndPrototype{}%
6618 \def\markdownRendererUlEndTightPrototype{}%
6619 \def\markdownRendererOlBeginPrototype{}%
6620 \def\markdownRendererOlBeginTightPrototype{}%
6621 \def\markdownRendererOlItemPrototype{}%
6622 \def\markdownRendererOlItemWithNumberPrototype#1{}%
6623 \def\markdownRendererOlItemEndPrototype{}%
6624 \def\markdownRendererOlEndPrototype{}%
6625 \def\markdownRendererOlEndTightPrototype{}%
6626 \def\markdownRendererDlBeginPrototype{}%
6627 \def\markdownRendererDlBeginTightPrototype{}%
6628 \def\markdownRendererDlItemPrototype#1{#1}%
6629 \def\markdownRendererDlItemEndPrototype{}%
6630 \def\markdownRendererDlDefinitionBeginPrototype{}%
6631 \def\markdownRendererDlDefinitionEndPrototype{\par}%
6632 \def\markdownRendererDlEndPrototype{}%
6633 \def\markdownRendererDlEndTightPrototype{}%
6634 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
6635 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
6636 \def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
6637 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
6638 \def\markdownRendererInputVerbatimPrototype#1{%
6639   \par{\tt\input#1\relax{}}\par}%
6640 \def\markdownRendererInputFencedCodePrototype#1#2{%
6641   \markdownRendererInputVerbatimPrototype{#1}}%
6642 \def\markdownRendererHeadingOnePrototype#1{#1}%
6643 \def\markdownRendererHeadingTwoPrototype#1{#1}%
6644 \def\markdownRendererHeadingThreePrototype#1{#1}%
6645 \def\markdownRendererHeadingFourPrototype#1{#1}%
6646 \def\markdownRendererHeadingFivePrototype#1{#1}%
6647 \def\markdownRendererHeadingSixPrototype#1{#1}%
6648 \def\markdownRendererHorizontalRulePrototype{}%
6649 \def\markdownRendererFootnotePrototype#1{#1}%
6650 \def\markdownRendererCitePrototype#1{}%
6651 \def\markdownRendererTextCitePrototype#1{}%
6652 \def\markdownRendererTickedBoxPrototype{[X]}%

```

```

6653 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
6654 \def\markdownRendererUntickedBoxPrototype{[ ]}%

```

**3.2.2.1 YAML Metadata Renderer Prototypes** To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_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  $p$ .

```

6655 \ExplSyntaxOn
6656 \seq_new:N \g_@@_jekyll_data_datatypes_seq
6657 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
6658 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
6659 \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.

```

6660 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
6661 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
6662 {
6663   \seq_if_empty:NF
6664     \g_@@_jekyll_data_datatypes_seq
6665     {
6666       \seq_get_right:NN
6667       \g_@@_jekyll_data_datatypes_seq
6668       \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.

```

6669     \str_if_eq:NNTF
6670     \l_tmpa_tl
6671     \c_@@_jekyll_data_sequence_tl
6672     {
6673     \seq_put_right:Nn

```



```

6674         \g_@@_jekyll_data_wildcard_absolute_address_seq
6675         { * }
6676     }
6677     {
6678         \seq_put_right:Nn
6679         \g_@@_jekyll_data_wildcard_absolute_address_seq
6680         { #1 }
6681     }
6682 }
6683 }

```

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.

```

6684 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
6685 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
6686 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
6687 {
6688     \seq_pop_left:NN #1 \l_tmpa_tl
6689     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
6690     \seq_put_left:NV #1 \l_tmpa_tl
6691 }
6692 \cs_new:Nn \markdown_jekyll_data_update_address_tls:

```

```

6693 {
6694   \markdown_jekyll_data_concatenate_address:NN
6695   \g_@@_jekyll_data_wildcard_absolute_address_seq
6696   \g_@@_jekyll_data_wildcard_absolute_address_tl
6697   \seq_get_right:NN
6698   \g_@@_jekyll_data_wildcard_absolute_address_seq
6699   \g_@@_jekyll_data_wildcard_relative_address_tl
6700 }

```

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.

```

6701 \cs_new:Nn \markdown_jekyll_data_push:nN
6702 {
6703   \markdown_jekyll_data_push_address_segment:n
6704   { #1 }
6705   \seq_put_right:NV
6706   \g_@@_jekyll_data_datatypes_seq
6707   #2
6708   \markdown_jekyll_data_update_address_tls:
6709 }
6710 \cs_new:Nn \markdown_jekyll_data_pop:
6711 {
6712   \seq_pop_right:NN
6713   \g_@@_jekyll_data_wildcard_absolute_address_seq
6714   \l_tmpa_tl
6715   \seq_pop_right:NN
6716   \g_@@_jekyll_data_datatypes_seq
6717   \l_tmpa_tl
6718   \markdown_jekyll_data_update_address_tls:
6719 }

```

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.

```

6720 \cs_new:Nn \markdown_jekyll_data_set_keyval:nN
6721 {
6722   \keys_set_known:nn
6723   { markdown/jekyllData }
6724   { { #1 } = { #2 } }
6725 }
6726 \cs_generate_variant:Nn
6727 \markdown_jekyll_data_set_keyval:nN
6728 { Vn }
6729 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
6730 {
6731   \markdown_jekyll_data_push:nN
6732   { #1 }
6733   \c_@@_jekyll_data_scalar_tl

```

```

6734 \markdown_jekyll_data_set_keyval:Vn
6735 \g_@@_jekyll_data_wildcard_absolute_address_tl
6736 { #2 }
6737 \markdown_jekyll_data_set_keyval:Vn
6738 \g_@@_jekyll_data_wildcard_relative_address_tl
6739 { #2 }
6740 \markdown_jekyll_data_pop:
6741 }

```

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

```

6742 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
6743 \markdown_jekyll_data_push:nN
6744 { #1 }
6745 \c_@@_jekyll_data_sequence_tl
6746 }
6747 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
6748 \markdown_jekyll_data_push:nN
6749 { #1 }
6750 \c_@@_jekyll_data_mapping_tl
6751 }
6752 \def\markdownRendererJekyllDataSequenceEndPrototype{
6753 \markdown_jekyll_data_pop:
6754 }
6755 \def\markdownRendererJekyllDataMappingEndPrototype{
6756 \markdown_jekyll_data_pop:
6757 }
6758 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
6759 \markdown_jekyll_data_set_keyvals:nn
6760 { #1 }
6761 { #2 }
6762 }
6763 \def\markdownRendererJekyllDataEmptyPrototype#1{}
6764 \def\markdownRendererJekyllDataNumberPrototype#1#2{
6765 \markdown_jekyll_data_set_keyvals:nn
6766 { #1 }
6767 { #2 }
6768 }
6769 \def\markdownRendererJekyllDataStringPrototype#1#2{
6770 \markdown_jekyll_data_set_keyvals:nn
6771 { #1 }
6772 { #2 }
6773 }
6774 \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 2.1.2).

```
6775 \ExplSyntaxOn
6776 \tl_new:N \g_@@_formatted_lua_options_tl
6777 \cs_new:Nn \@@_format_lua_options:
6778 {
6779   \tl_gclear:N
6780     \g_@@_formatted_lua_options_tl
6781   \seq_map_function:NN
6782     \g_@@_lua_options_seq
6783     \@@_format_lua_option:n
6784 }
6785 \cs_new:Nn \@@_format_lua_option:n
6786 {
6787   \@@_typecheck_option:n
6788     { #1 }
6789   \@@_get_option_type:nN
6790     { #1 }
6791   \l_tmpa_tl
6792   \bool_if:nTF
6793     {
6794       \str_if_eq_p:VV
6795         \l_tmpa_tl
6796         \c_@@_option_type_boolean_tl ||
6797       \str_if_eq_p:VV
6798         \l_tmpa_tl
6799         \c_@@_option_type_number_tl ||
6800       \str_if_eq_p:VV
6801         \l_tmpa_tl
6802         \c_@@_option_type_counter_tl
6803     }
6804   {
6805     \@@_get_option_value:nN
6806       { #1 }
6807     \l_tmpa_tl
6808     \tl_gput_right:Nx
6809       \g_@@_formatted_lua_options_tl
6810       { #1~::~ \l_tmpa_tl ,~ }
6811   }
6812   {
6813     \@@_get_option_value:nN
6814       { #1 }
6815     \l_tmpa_tl
6816     \tl_gput_right:Nx
```

```

6817         \g_@@_formatted_lua_options_tl
6818         { #1~== " \l_tmpa_tl " ,~ }
6819     }
6820 }
6821 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
6822 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
6823 \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.

```

6824 \def\markdownPrepare{%

```

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

```

6825     local lfs = require("lfs")
6826     local cacheDir = "\markdownOptionCacheDir"
6827     if not lfs.isdir(cacheDir) then
6828         assert(lfs.mkdir(cacheDir))
6829     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.

```

6830     local md = require("markdown")
6831     local convert = md.new(\markdownLuaOptions)
6832 }%

```

### 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.

```

6833 \ExplSyntaxOn
6834 \cs_new:Nn
6835   \@@_if_option:nTF
6836   {
6837     \@@_get_option_type:nN
6838     { #1 }
6839     \l_tmpa_tl
6840     \str_if_eq:NNF
6841     \l_tmpa_tl
6842     \c_@@_option_type_boolean_tl
6843     {
6844       \msg_error:nxxx
6845       { @@ }
6846       { expected-boolean-option }
6847       { #1 }
6848       { \l_tmpa_tl }

```

```

6849     }
6850     \@@_get_option_value:nN
6851     { #1 }
6852     \l_tmpa_tl
6853     \str_if_eq:NNTF
6854     \l_tmpa_tl
6855     \c_@@_option_value_true_tl
6856     { #2 }
6857     { #3 }
6858   }
6859 \msg_new:nnn
6860 { @@ }
6861 { expected-boolean-option }
6862 {
6863   Option~#1~has~type~#2,~
6864   but~a~boolean~was~expected.
6865 }
6866 \let\markdownIfOption=\@@_if_option:nTF
6867 \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.

```

6868 \csname newread\endcsname\markdownInputFileStream
6869 \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

6870 \begingroup
6871 \catcode\^^I=12%
6872 \gdef\markdownReadAndConvertTab{^^I}%
6873 \endgroup

```

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

```

6874 \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.

```

6875 \catcode\^^M=13%
6876 \catcode\^^I=13%
6877 \catcode`|=0%
6878 \catcode`\=12%
6879 |catcode`@=14%
6880 |catcode`|=12@
6881 |gdef|markdownReadAndConvert#1#2{#@

```

```
6882 |begingroup@
```

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

```
6883 |markdownIfOption{frozenCache}{-}{@
6884 |immediate|openout|markdownOutputFileStream@
6885 |markdownOptionInputTempFileName|relax@
6886 |markdownInfo{Buffering markdown input into the temporary @
6887 |input file "|markdownOptionInputTempFileName" and scanning @
6888 |for the closing token sequence "#1"}@
6889 }@
```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
6890 |def|do##1{|catcode`##1=12}|dospecials@
6891 |catcode`| =12@
6892 |markdownMakeOther@
```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `\markdownOptionStripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols ( $\sim$ M) are produced.

```
6893 |def|markdownReadAndConvertStripPercentSign##1{@
6894 |markdownIfOption{stripPercentSigns}{@
6895 |if##1%@
6896 |expandafter|expandafter|expandafter@
6897 |markdownReadAndConvertProcessLine@
6898 |else@
6899 |expandafter|expandafter|expandafter@
6900 |markdownReadAndConvertProcessLine@
6901 |expandafter|expandafter|expandafter##1@
6902 |fi@
6903 }{@
6904 |expandafter@
6905 |markdownReadAndConvertProcessLine@
6906 |expandafter##1@
6907 }@
6908 }@
```

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

```
6909 |def|markdownReadAndConvertProcessLine##1##2##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.

```
6910     |ifx|relax##3|relax@
6911         |markdownIfOption{frozenCache}{-}{@
6912             |immediate|write|markdownOutputFileStream{##1}@
6913         }@
6914     |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.

```
6915     |def^^M{@
6916         |markdownInfo{The ending token sequence was found}@
6917         |markdownIfOption{frozenCache}{-}{@
6918             |immediate|closeout|markdownOutputFileStream@
6919         }@
6920     |endgroup@
6921     |markdownInput{@
6922         |markdownOptionOutputDir@
6923         /|markdownOptionInputTempFileName@
6924     }@
6925     #2}@
6926 |fi@
```

Repeat with the next line.

```
6927     ^^M}@
```

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

```
6928     |catcode`|^I=13@
6929     |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.

```
6930     |catcode`|^M=13@
6931     |def^^M##1^^M{@
6932         |def^^M###1^^M{@
6933             |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
6934         ^^M}@
6935     ^^M}@
```

Reset the character categories back to the former state.

```
6936 |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.

```

6937 \ExplSyntaxOn
6938 \int_compare:nT
6939   { \markdownMode = 3 }
6940   {
6941     \markdownInfo{Using mode~3:~The~lt3luabridge~package}
6942     \file_input:n { lt3luabridge.tex }
6943     \cs_new:Npn
6944       \markdownLuaExecute
6945       { \luabridgeExecute }
6946   }
6947 \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 3.1.1].

```

6948 \ifnum\markdownMode<2\relax
6949 \ifnum\markdownMode=0\relax
6950   \markdownWarning{Using mode 0: Shell escape via write18
6951                   (deprecated, to be removed in Markdown 3.0.0)}%
6952 \else
6953   \markdownWarning{Using mode 1: Shell escape via os.execute
6954                   (deprecated, to be removed in Markdown 3.0.0)}%
6955 \fi

```

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

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

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

```

6956 \ifx\pdfshellescape\undefined
6957   \ifx\shellescape\undefined
6958     \ifnum\markdownMode=0\relax
6959       \def\markdownExecuteShellEscape{1}%
6960     \else

```

```

6961     \def\markdownExecuteShellEscape{%
6962         \directlua{tex.sprint(status.shell_escape or "1")}}%
6963     \fi
6964 \else
6965     \let\markdownExecuteShellEscape\shellescape
6966 \fi
6967 \else
6968     \let\markdownExecuteShellEscape\pdfshellescape
6969 \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.

```

6970 \ifnum\markdownMode=0\relax
6971     \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
6972 \else
6973     \def\markdownExecuteDirect#1{%
6974         \directlua{os.execute("\luaescapestring{#1}")}}%
6975 \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.

```

6976 \def\markdownExecute#1{%
6977     \ifnum\markdownExecuteShellEscape=1\relax
6978         \markdownExecuteDirect{#1}%
6979     \else
6980         \markdownError{I can not access the shell}{Either run the TeX
6981             compiler with the --shell-escape or the --enable-write18 flag,
6982             or set shell_escape=t in the texmf.cnf file}%
6983     \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.

```

6984 \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.

```

6985     \catcode`\|=0%
6986     \catcode`\|=12%
6987     \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.

```

6988         \immediate\openout\markdownOutputFileStream=%
6989         \markdownOptionHelperScriptFileName

```

```

6990 |markdownInfo{Writing a helper Lua script to the file
6991   "|markdownOptionHelperScriptFileName"}%
6992 |immediate|write|markdownOutputFileStream{%
6993   local ran_ok, error = pcall(function()
6994     local ran_ok, kpse = pcall(require, "kpse")
6995     if ran_ok then kpse.set_program_name("luatex") end
6996     #1
6997   end)

```

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

```

6998   if not ran_ok then
6999     local file = io.open("%
7000     |markdownOptionOutputDir
7001     /|markdownOptionErrorTempFileName", "w")
7002     if file then
7003       file:write(error .. "\n")
7004       file:close()
7005     end
7006     print('\|markdownError{An error was encountered while executing
7007           Lua code}{For further clues, examine the file
7008           "|markdownOptionOutputDir
7009           /|markdownOptionErrorTempFileName"}')
7010   end}%
7011 |immediate|closeout|markdownOutputFileStream

```

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

```

7012 |markdownInfo{Executing a helper Lua script from the file
7013   "|markdownOptionHelperScriptFileName" and storing the result in the
7014   file "|markdownOptionOutputTempFileName"}%
7015 |markdownExecute{texlua "|markdownOptionOutputDir
7016   /|markdownOptionHelperScriptFileName" > %
7017   "|markdownOptionOutputDir
7018   /|markdownOptionOutputTempFileName"}%

```

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

```

7019   |input|markdownOptionOutputTempFileName|relax}%
7020 |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.

```

7021 \fi
7022 \ifnum\markdownMode=2\relax
7023   \markdownWarning{Using mode 2: Direct Lua access
7024     (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,

```
7025 \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.

```

7026   \catcode`\|=0%
7027   \catcode`\|=12%
7028   |gdef|markdownLuaExecute#1{%
7029     |directlua{%
7030       local function print(input)
7031         local output = {}
7032         for line in input:gmatch("[^\r\n]+") do
7033           table.insert(output, line)
7034         end
7035         tex.print(output)
7036       end
7037       #1
7038     }%
7039   }%
7040 |endgroup
7041 \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.

```
7042 \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.

```

7043   \catcode`\|=0%
7044   \catcode`\|=12%
7045   |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.

```

7046     |begingroup
7047     |catcode`\|=12

```

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

```

7048 |markdownIfOption{frozenCache}{%
7049 |  ifnum|markdownOptionFrozenCacheCounter=0|relax
7050 |  |markdownInfo{Reading frozen cache from
7051 |    "|markdownOptionFrozenCacheFileName"}%
7052 |  |input|markdownOptionFrozenCacheFileName|relax
7053 |  |fi
7054 |  |markdownInfo{Including markdown document number
7055 |    "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
7056 |  |curname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcurname
7057 |  |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7058 |}%
7059 |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.

```

7060 |openin|markdownInputFileStream#1
7061 |closein|markdownInputFileStream
7062 |markdownPrepareLuaOptions
7063 |markdownLuaExecute{%
7064 |  |markdownPrepare
7065 |  local file = assert(io.open("#1", "r"),
7066 |    [[could not open file "#1" for reading]])
7067 |  local input = assert(file:read("*a"))
7068 |  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.

```

7069 |  print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%

```

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

```

7070 |  |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7071 |}%
7072 |endgroup
7073 |}%
7074 |endgroup

```

### 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 [10, Section 9].

As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

7075 | \def\markdownVersionSpace{ }%
7076 | \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
7077 | \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.

```
7078 \ExplSyntaxOn
7079 \@@_latex_define_renderers:
7080 \@@_latex_define_renderer_prototypes:
7081 \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).

```
7082 \let\markdownInputPlainTeX\markdownInput
7083 \renewcommand\markdownInput[2] [] {%
7084   \begingroup
7085     \markdownSetup{#1}%
7086     \markdownInputPlainTeX{#2}%
7087   \endgroup}%
```

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

```
7088 \renewenvironment{markdown}{%
7089   \markdownReadAndConvert@markdown{}}{%
7090   \markdownEnd}%
7091 \renewenvironment{markdown*}[1]{%
7092   \markdownSetup{#1}%
7093   \markdownReadAndConvert@markdown*}{%
7094   \markdownEnd}%
7095 \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*.

```
7096   \catcode`\|=0\catcode`\<=1\catcode`\>=2%
7097   \catcode`\|=12\catcode`\{=12\catcode`\}=12%
7098   |gdef|markdownReadAndConvert@markdown#1<%
7099     |markdownReadAndConvert<\end{markdown#1}>%
7100     <|end<markdown#1>>%
7101 |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.

```
7102 \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.

```
7103 \newcommand\markdownLaTeXThemeName{}
7104 \seq_new:N \g_@@_latex_themes_seq
7105 \seq_put_right:NV
7106   \g_@@_latex_themes_seq
7107   \markdownLaTeXThemeName
7108 \newcommand\markdownLaTeXThemeLoad[2]{
7109   \def\@tempa{%
7110     \def\markdownLaTeXThemeName{#2}
7111     \seq_put_right:NV
7112       \g_@@_latex_themes_seq
7113       \markdownLaTeXThemeName
7114     \RequirePackage{#1}
7115     \seq_pop_right:NN
7116       \g_@@_latex_themes_seq
7117     \l_tmpa_tl
7118     \seq_get_right:NN
7119       \g_@@_latex_themes_seq
7120     \l_tmpa_tl
7121     \exp_args:NNV
7122     \def
7123       \markdownLaTeXThemeName
7124     \l_tmpa_tl}
7125 \ifmarkdownLaTeXLoaded
7126   \@tempa
7127 \else
7128   \exp_args:No
7129     \AtEndOfPackage
7130   { \@tempa }
7131 \fi}
7132 \ExplSyntaxOff
```

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

```
7133 \markdownSetup{fencedCode}%
```

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

```
7134 \RequirePackage{ifthen,grffile}
```

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

```
7135 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
7136 \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:

```

7137 \renewcommand\markdownRendererInputFencedCode[2]{%
7138   \def\next##1 ##2\relax{%
7139     \ifthenelse{\equal{##1}{dot}}{%
7140       \markdownIfOption{frozenCache}{}{%
7141         \immediate\write18{%
7142           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
7143           then
7144             dot -Tpdf -o #1.pdf #1;
7145             cp #1 #1.pdf.source;
7146             fi}}%

```

We include the typeset image using the image token renderer:

```

7147   \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:

```

7148   }{%
7149   \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
7150   }%
7151 }%
7152 \next#2 \relax}%

```

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

```

7153 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
7154 \markdownRendererImagePrototype

```

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

```

7155 \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.

```

7156 \newcount\markdown@witiko@graphicx@http@counter
7157 \markdown@witiko@graphicx@http@counter=0
7158 \newcommand\markdown@witiko@graphicx@http@filename{%
7159   \markdownOptionCacheDir/witiko_graphicx_http%
7160   .\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.

```

7161 \newcommand\markdown@witiko@graphicx@http@download[2]{%
7162   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:

```
7163 \begingroup
7164 \catcode`\%=12
7165 \catcode`\^^A=14
```

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

```
7166 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
7167   \begingroup
7168     \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:

```
7169     \markdownIfOption{frozenCache}{}{^^A
7170       \immediate\write18{^^A
7171         mkdir -p "\markdownOptionCacheDir";
7172         if printf '%s' "#3" | grep -q -E '^https?:';
7173         then
```

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

```
7174         OUTPUT_PREFIX="\markdownOptionCacheDir";
7175         OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
7176         OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
7177         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

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

```
7178         if ! [ -e "$OUTPUT" ];
7179         then
7180           \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
7181           printf '%s' "$OUTPUT" > "\filename";
7182         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:

```
7183         else
7184           printf '%s' '#3' > "\filename";
7185         fi}}^^A
```

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

```
7186     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
7187     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
7188     {#1}{#2}{\filename}{#4}^^A
7189   \endgroup
7190   \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
7191 \endgroup
```

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

```
7192 \renewcommand\markdownRendererTildePrototype{~}%
```

### 3.3.3 Options

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

```
7193 \DeclareOption*{%
7194   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
7195 \ProcessOptions\relax
```

After processing the options, activate the `renderers`, `rendererPrototypes`, and `code` keys. The `code` key is used to immediately expand and execute code, which can be especially useful in  $\text{\LaTeX}$  setup snippets.

```
7196 \ExplSyntaxOn
7197 \keys_define:nn
7198   { markdown/latex-options }
7199   {
7200     renderers .code:n = {
7201       \keys_set:nn
7202         { markdown/latex-options/renderers }
7203         { #1 }
7204     },
7205     rendererPrototypes .code:n = {
7206       \keys_set:nn
7207         { markdown/latex-options/renderer-prototypes }
7208         { #1 }
7209     },
7210     code .code:n = { #1 },
7211   }
7212 \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.

```
7213 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

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

```
7214 \markdownIfOption{tightLists}{
7215   \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}%
7216 }{}
```

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.

```

7217 \@ifpackageloaded{paralist}{
7218   \markdownSetup{rendererPrototypes={
7219     ulBeginTight = {\begin{compactitem}},
7220     ulEndTight = {\end{compactitem}},
7221     olBeginTight = {\begin{compactenum}},
7222     olEndTight = {\end{compactenum}},
7223     dlBeginTight = {\begin{compactdesc}},
7224     dlEndTight = {\end{compactdesc}}}}
7225   }{
7226   \markdownSetup{rendererPrototypes={
7227     ulBeginTight = {\markdownRendererUlBegin},
7228     ulEndTight = {\markdownRendererUlEnd},
7229     olBeginTight = {\markdownRendererOlBegin},
7230     olEndTight = {\markdownRendererOlEnd},
7231     dlBeginTight = {\markdownRendererDlBegin},
7232     dlEndTight = {\markdownRendererDlEnd}}}}
7233 \RequirePackage{amsmath,ifthen}

```

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

```

7234 \@ifpackageloaded{unicode-math}{
7235   \markdownSetup{rendererPrototypes={
7236     untickedBox = {\$ \mdlgwhtsquare$},
7237   }}
7238 }{
7239   \RequirePackage{amssymb}
7240   \markdownSetup{rendererPrototypes={
7241     untickedBox = {\$ \square$},
7242   }}
7243 }
7244 \RequirePackage{csvsimple}
7245 \RequirePackage{fancyvrb}
7246 \RequirePackage{graphicx}
7247 \markdownSetup{rendererPrototypes={
7248   lineBreak = {\},
7249   leftBrace = {\textbraceleft},
7250   rightBrace = {\textbraceright},
7251   dollarSign = {\textdollar},
7252   underscore = {\textunderscore},
7253   circumflex = {\textasciicircum},
7254   backslash = {\textbackslash},
7255   tilde = {\textasciitilde},
7256   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\TeX$  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.

```

7257 codeSpan = {%
7258   \ifmmode
7259     \text{#1}%
7260   \else
7261     \texttt{#1}%
7262   \fi
7263 },
7264 contentBlock = {%
7265   \ifthenelse{\equal{#1}{csv}}{%
7266     \begin{table}%
7267       \begin{center}%
7268         \csvautotabular{#3}%
7269       \end{center}
7270     \ifx\empty#4\empty\else
7271       \caption{#4}%
7272     \fi
7273   \end{table}%
7274 }{%
7275   \ifthenelse{\equal{#1}{tex}}{%
7276     \catcode`\%=14\relax
7277     \input #3\relax
7278     \catcode`\%=12\relax
7279   }{%
7280     \markdownInput{#3}%
7281   }%
7282 }%
7283 },
7284 image = {%
7285   \begin{figure}%
7286     \begin{center}%
7287       \includegraphics{#3}%
7288     \end{center}%
7289     \ifx\empty#4\empty\else
7290       \caption{#4}%
7291     \fi
7292   \end{figure}},
7293 ulBegin = {\begin{itemize}},
7294 ulEnd = {\end{itemize}},
7295 olBegin = {\begin{enumerate}},

```

---

<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.

```

7296 olItem = {\item{}},
7297 olItemWithNumber = {\item[#1.]},
7298 olEnd = {\end{enumerate}},
7299 dlBegin = {\begin{description}},
7300 dlItem = {\item[#1]},
7301 dlEnd = {\end{description}},
7302 emphasis = {\emph{#1}},
7303 tickedBox = {\$\boxtimes$},
7304 halfTickedBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

7305 headerAttributeContextBegin = {
7306   \markdownSetup{
7307     rendererPrototypes = {
7308       attributeIdentifier = {%
7309         \begingroup
7310         \def\next####1{%
7311           \def####1#####1{%
7312             \endgroup
7313             ####1{#####1}%
7314             \label{##1}%
7315           }%
7316         }%
7317         \next\markdownRendererHeadingOne
7318         \next\markdownRendererHeadingTwo
7319         \next\markdownRendererHeadingThree
7320         \next\markdownRendererHeadingFour
7321         \next\markdownRendererHeadingFive
7322         \next\markdownRendererHeadingSix
7323       },
7324     },
7325   }%
7326 },
7327 blockQuoteBegin = {\begin{quotation}},
7328 blockQuoteEnd = {\end{quotation}},
7329 inputVerbatim = {\VerbatimInput{#1}},
7330 inputFencedCode = {%
7331   \ifx\relax#2\relax
7332     \VerbatimInput{#1}%
7333   \else
7334     \@ifundefined{minted@code}{%
7335       \@ifundefined{lst@version}{%
7336         \markdownRendererInputFencedCode{#1}{}%

```

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

```

7337   }{%
7338     \lstinputlisting[language=#2]{#1}%

```

```
7339     }%
```

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

```
7340     }{%
7341     \inputminted{#2}{#1}%
7342     }%
7343     \fi},
7344     horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
7345     footnote = {\footnote{#1}}}}
```

Support the nesting of strong emphasis.

```
7346 \ExplSyntaxOn
7347 \def\markdownLATEXStrongEmphasis#1{%
7348   \str_if_in:NnTF
7349     \f@series
7350     { b }
7351     { \textnormal{#1} }
7352     { \textbf{#1} }
7353 }
7354 \ExplSyntaxOff
7355 \markdownSetup{rendererPrototypes={strongEmphasis={%
7356   \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```
7357 \@ifundefined{chapter}{%
7358   \markdownSetup{rendererPrototypes = {
7359     headingOne = {\section{#1}},
7360     headingTwo = {\subsection{#1}},
7361     headingThree = {\subsubsection{#1}},
7362     headingFour = {\paragraph{#1}\leavevmode},
7363     headingFive = {\subparagraph{#1}\leavevmode}}}
7364 }{%
7365   \markdownSetup{rendererPrototypes = {
7366     headingOne = {\chapter{#1}},
7367     headingTwo = {\section{#1}},
7368     headingThree = {\subsection{#1}},
7369     headingFour = {\subsubsection{#1}},
7370     headingFive = {\paragraph{#1}\leavevmode},
7371     headingSix = {\subparagraph{#1}\leavevmode}}}
7372 }%
```

**3.3.4.1 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```
7373 \markdownSetup{
7374   rendererPrototypes = {
7375     ulItem = {%
```

```

7376     \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
7377   },
7378 },
7379 }
7380 \def\markdownLaTeXUItem{%
7381   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
7382     \item[\markdownLaTeXCheckbox]%
7383     \expandafter\@gobble
7384   \else
7385     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
7386       \item[\markdownLaTeXCheckbox]%
7387       \expandafter\expandafter\expandafter\@gobble
7388     \else
7389       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
7390         \item[\markdownLaTeXCheckbox]%
7391         \expandafter\expandafter\expandafter\expandafter
7392         \expandafter\expandafter\expandafter\@gobble
7393       \else
7394         \item{}%
7395       \fi
7396     \fi
7397   \fi
7398 }

```

**3.3.4.2 HTML elements** If the `html` option is enabled and we are using `TeX4ht`<sup>9</sup>, we will pass HTML elements to the output HTML document unchanged.

```

7399 \@ifundefined{HCode}{-}{-}
7400 \markdownSetup{
7401   rendererPrototypes = {
7402     inlineHtmlTag = {%
7403       \ifvmode
7404         \IgnorePar
7405       \EndP
7406     \fi
7407     \HCode{#1}%
7408   },
7409   inputBlockHtmlElement = {%
7410     \ifvmode
7411       \IgnorePar
7412     \fi
7413     \EndP
7414     \special{t4ht*<#1}%
7415     \par
7416     \ShowPar
7417   },

```

---

<sup>9</sup>See <https://tug.org/tex4ht/>.

```

7418     },
7419   }
7420 }

```

**3.3.4.3 Citations** Here is a basic implementation for citations that uses the  $\text{\LaTeX}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

7421 \newcount\markdownLaTeXCitationsCounter
7422
7423 % Basic implementation
7424 \RequirePackage{gobble}
7425 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
7426   \advance\markdownLaTeXCitationsCounter by 1\relax
7427   \ifx\relax#4\relax
7428     \ifx\relax#5\relax
7429       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7430         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
7431         \expandafter\expandafter\expandafter
7432         \expandafter\expandafter\expandafter\expandafter
7433         \@gobblethree
7434       \fi
7435     \else% Before a postnote (#5), dump the accumulator
7436       \ifx\relax#1\relax\else
7437         \cite{#1}%
7438       \fi
7439       \cite[#5]{#6}%
7440       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7441       \else
7442         \expandafter\expandafter\expandafter
7443         \expandafter\expandafter\expandafter\expandafter
7444         \expandafter\expandafter\expandafter
7445         \expandafter\expandafter\expandafter\expandafter
7446         \markdownLaTeXBasicCitations
7447       \fi
7448       \expandafter\expandafter\expandafter
7449       \expandafter\expandafter\expandafter\expandafter{%
7450       \expandafter\expandafter\expandafter
7451       \expandafter\expandafter\expandafter\expandafter}%
7452       \expandafter\expandafter\expandafter
7453       \expandafter\expandafter\expandafter\expandafter{%
7454       \expandafter\expandafter\expandafter
7455       \expandafter\expandafter\expandafter\expandafter}%
7456       \expandafter\expandafter\expandafter
7457       \@gobblethree
7458     \fi

```



```

7459 \else% Before a prenote (#4), dump the accumulator
7460 \ifx\relax#1\relax\else
7461 \cite{#1}%
7462 \fi
7463 \ifnum\markdownLaTeXCitationsCounter>1\relax
7464 \space % Insert a space before the prenote in later citations
7465 \fi
7466 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
7467 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7468 \else
7469 \expandafter\expandafter\expandafter
7470 \expandafter\expandafter\expandafter\expandafter
7471 \markdownLaTeXBasicCitations
7472 \fi
7473 \expandafter\expandafter\expandafter{%
7474 \expandafter\expandafter\expandafter}%
7475 \expandafter\expandafter\expandafter{%
7476 \expandafter\expandafter\expandafter}%
7477 \expandafter
7478 \@gobblethree
7479 \fi\markdownLaTeXBasicCitations{#1#2#6},)
7480 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
7481
7482 % Natbib implementation
7483 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
7484 \advance\markdownLaTeXCitationsCounter by 1\relax
7485 \ifx\relax#3\relax
7486 \ifx\relax#4\relax
7487 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7488 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
7489 \expandafter\expandafter\expandafter
7490 \expandafter\expandafter\expandafter\expandafter
7491 \@gobbletwo
7492 \fi
7493 \else% Before a postnote (#4), dump the accumulator
7494 \ifx\relax#1\relax\else
7495 \citep{#1}%
7496 \fi
7497 \citep[] [#4]{#5}%
7498 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7499 \else
7500 \expandafter\expandafter\expandafter
7501 \expandafter\expandafter\expandafter\expandafter
7502 \expandafter\expandafter\expandafter
7503 \expandafter\expandafter\expandafter\expandafter
7504 \markdownLaTeXNatbibCitations
7505 \fi

```

```

7506     \expandafter\expandafter\expandafter
7507     \expandafter\expandafter\expandafter\expandafter{%
7508     \expandafter\expandafter\expandafter
7509     \expandafter\expandafter\expandafter\expandafter}%
7510     \expandafter\expandafter\expandafter
7511     \@gobbletwo
7512     \fi
7513 \else% Before a prenote (#3), dump the accumulator
7514     \ifx\relax#1\relax\relax\else
7515         \citep{#1}%
7516     \fi
7517     \citep[#3][#4]{#5}%
7518     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7519     \else
7520         \expandafter\expandafter\expandafter
7521         \expandafter\expandafter\expandafter\expandafter
7522         \markdownLaTeXNatbibCitations
7523     \fi
7524     \expandafter\expandafter\expandafter{%
7525     \expandafter\expandafter\expandafter}%
7526     \expandafter
7527     \@gobbletwo
7528     \fi\markdownLaTeXNatbibCitations{#1,#5}}
7529 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
7530 \advance\markdownLaTeXCitationsCounter by 1\relax
7531 \ifx\relax#3\relax
7532     \ifx\relax#4\relax
7533         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7534             \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
7535             \expandafter\expandafter\expandafter
7536             \expandafter\expandafter\expandafter\expandafter
7537             \@gobbletwo
7538         \fi
7539     \else% After a prenote or a postnote, dump the accumulator
7540         \ifx\relax#1\relax\relax\else
7541             \citet{#1}%
7542         \fi
7543         , \citet[#3][#4]{#5}%
7544         \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
7545         ,
7546         \else
7547             \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
7548             ,
7549         \fi
7550     \fi
7551     \expandafter\expandafter\expandafter
7552     \expandafter\expandafter\expandafter\expandafter

```

```

7553     \markdownLaTeXNatbibTextCitations
7554     \expandafter\expandafter\expandafter
7555     \expandafter\expandafter\expandafter\expandafter{%
7556     \expandafter\expandafter\expandafter
7557     \expandafter\expandafter\expandafter\expandafter}%
7558     \expandafter\expandafter\expandafter
7559     \@gobbletwo
7560     \fi
7561 \else% After a prenote or a postnote, dump the accumulator
7562     \ifx\relax#1\relax\relax\else
7563         \citet{#1}%
7564     \fi
7565     , \citet[#3][#4]{#5}%
7566     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
7567     ,
7568     \else
7569         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
7570     ,
7571     \fi
7572     \fi
7573     \expandafter\expandafter\expandafter
7574     \markdownLaTeXNatbibTextCitations
7575     \expandafter\expandafter\expandafter{%
7576     \expandafter\expandafter\expandafter}%
7577     \expandafter
7578     \@gobbletwo
7579     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
7580
7581 % BibLaTeX implementation
7582 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
7583     \advance\markdownLaTeXCitationsCounter by 1\relax
7584     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7585         \autocites#1[#3][#4]{#5}%
7586         \expandafter\@gobbletwo
7587     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
7588 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
7589     \advance\markdownLaTeXCitationsCounter by 1\relax
7590     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7591         \textcites#1[#3][#4]{#5}%
7592         \expandafter\@gobbletwo
7593     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
7594
7595 \markdownSetup{rendererPrototypes = {
7596     cite = {%
7597         \markdownLaTeXCitationsCounter=1%
7598         \def\markdownLaTeXCitationsTotal{#1}%
7599         \@ifundefined{autocites}{%

```

```

7600     \@ifundefined{citep}{%
7601         \expandafter\expandafter\expandafter
7602         \markdownLaTeXBasicCitations
7603         \expandafter\expandafter\expandafter{%
7604         \expandafter\expandafter\expandafter}%
7605         \expandafter\expandafter\expandafter{%
7606         \expandafter\expandafter\expandafter}%
7607     }{%
7608         \expandafter\expandafter\expandafter
7609         \markdownLaTeXNatbibCitations
7610         \expandafter\expandafter\expandafter{%
7611         \expandafter\expandafter\expandafter}%
7612     }%
7613 }{%
7614     \expandafter\expandafter\expandafter
7615     \markdownLaTeXBibLaTeXCitations
7616     \expandafter{\expandafter}%
7617 }},
7618 textCite = {%
7619     \markdownLaTeXCitationsCounter=1%
7620     \def\markdownLaTeXCitationsTotal{#1}%
7621     \@ifundefined{autocites}{%
7622         \@ifundefined{citep}{%
7623             \expandafter\expandafter\expandafter
7624             \markdownLaTeXBasicTextCitations
7625             \expandafter\expandafter\expandafter{%
7626             \expandafter\expandafter\expandafter}%
7627             \expandafter\expandafter\expandafter{%
7628             \expandafter\expandafter\expandafter}%
7629         }{%
7630             \expandafter\expandafter\expandafter
7631             \markdownLaTeXNatbibTextCitations
7632             \expandafter\expandafter\expandafter{%
7633             \expandafter\expandafter\expandafter}%
7634         }%
7635     }{%
7636         \expandafter\expandafter\expandafter
7637         \markdownLaTeXBibLaTeXTextCitations
7638         \expandafter{\expandafter}%
7639     }}}

```

**3.3.4.4 Links** Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

7640 \RequirePackage{url}
7641 \RequirePackage{expl3}

```

```

7642 \ExplSyntaxOn
7643 \def\markdownRendererLinkPrototype{
7644   \begingroup
7645   \catcode`\#=12
7646   \def\next##1##2##3##4{
7647     \endgroup

```

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.

```

7648   \tl_set:Nn \l_tmpa_tl { ##1 }
7649   \tl_set:Nn \l_tmpb_tl { ##2 }
7650   \bool_set:Nn
7651     \l_tmpa_bool
7652     {
7653       \tl_if_eq_p:NN
7654         \l_tmpa_tl
7655         \l_tmpb_tl
7656     }
7657   \tl_set:Nn \l_tmpa_tl { ##4 }
7658   \bool_set:Nn
7659     \l_tmpb_bool
7660     {
7661       \tl_if_empty_p:N
7662         \l_tmpa_tl
7663     }
7664   \bool_if:nTF
7665     {
7666       \l_tmpa_bool && \l_tmpb_bool
7667     }
7668     {
7669       \markdownLaTeXRendererAutolink { ##2 } { ##3 }
7670     }{
7671       \markdownLaTeXRendererDirectOrIndirectLink { ##1 } { ##2 } { ##3 } { ##4 }
7672     }
7673   }
7674   \next
7675 }
7676 \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.

```

7677   \tl_set:Nn
7678     \l_tmpa_tl
7679     { #2 }
7680   \tl_trim_spaces:N
7681     \l_tmpa_tl
7682   \tl_set:Nx

```

```

7683   \l_tmpb_tl
7684   {
7685     \tl_range:Nnn
7686     \l_tmpa_tl
7687     { 1 }
7688     { 1 }
7689   }
7690   \str_if_eq:NNTF
7691   \l_tmpb_tl
7692   \c_hash_str
7693   {
7694     \tl_set:Nx
7695     \l_tmpb_tl
7696     {
7697       \tl_range:Nnn
7698       \l_tmpa_tl
7699       { 2 }
7700       { -1 }
7701     }
7702     \exp_args:NV
7703     \ref
7704     \l_tmpb_tl
7705   }{
7706     \url { #2 }
7707   }
7708 }
7709 \ExplSyntaxOff
7710 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
7711   #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.

```

7712 \newcount\markdownLaTeXRowCount
7713 \newcount\markdownLaTeXRowTotal
7714 \newcount\markdownLaTeXColumnCounter
7715 \newcount\markdownLaTeXColumnTotal
7716 \newtoks\markdownLaTeXTable
7717 \newtoks\markdownLaTeXTableAlignment
7718 \newtoks\markdownLaTeXTableEnd
7719 \AtBeginDocument{%
7720   \@ifpackageloaded{booktabs}{%
7721     \def\markdownLaTeXTopRule{\toprule}%
7722     \def\markdownLaTeXMidRule{\midrule}%
7723     \def\markdownLaTeXBottomRule{\bottomrule}%
7724   }{%
7725     \def\markdownLaTeXTopRule{\hline}%

```

```

7726     \def\markdownLaTeXMidRule{\hline}%
7727     \def\markdownLaTeXBottomRule{\hline}%
7728   }%
7729 }
7730 \markdownSetup{rendererPrototypes={
7731   table = {%
7732     \markdownLaTeXTable={}%
7733     \markdownLaTeXTableAlignment={}%
7734     \markdownLaTeXTableEnd={%
7735       \markdownLaTeXBottomRule
7736       \end{tabular}}}%
7737     \ifx\empty#1\empty\else
7738       \addto@hook\markdownLaTeXTable{%
7739         \begin{table}
7740         \centering}%
7741       \addto@hook\markdownLaTeXTableEnd{%
7742         \caption{#1}
7743         \end{table}}}%
7744     \fi
7745     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
7746     \markdownLaTeXRowCount=0%
7747     \markdownLaTeXRowTotal=#2%
7748     \markdownLaTeXColumnTotal=#3%
7749     \markdownLaTeXRenderTableRow
7750   }
7751 }}
7752 \def\markdownLaTeXRenderTableRow#1{%
7753   \markdownLaTeXColumnCounter=0%
7754   \ifnum\markdownLaTeXRowCount=0\relax
7755     \markdownLaTeXReadAlignments#1%
7756     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
7757       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
7758         \the\markdownLaTeXTableAlignment}}}%
7759     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
7760   \else
7761     \markdownLaTeXRenderTableCell#1%
7762     \fi
7763     \ifnum\markdownLaTeXRowCount=1\relax
7764       \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
7765     \fi
7766     \advance\markdownLaTeXRowCount by 1\relax
7767     \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
7768       \the\markdownLaTeXTable
7769       \the\markdownLaTeXTableEnd
7770       \expandafter\@gobble
7771     \fi\markdownLaTeXRenderTableRow}
7772 \def\markdownLaTeXReadAlignments#1{%

```

```

7773 \advance\markdownLaTeXColumnCounter by 1\relax
7774 \if#1d%
7775   \addto@hook\markdownLaTeXTableAlignment{1}%
7776 \else
7777   \addto@hook\markdownLaTeXTableAlignment{#1}%
7778 \fi
7779 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
7780   \expandafter\@gobble
7781 \fi\markdownLaTeXReadAlignments}
7782 \def\markdownLaTeXRenderTableCell#1{%
7783   \advance\markdownLaTeXColumnCounter by 1\relax
7784   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
7785     \addto@hook\markdownLaTeXTable{#1&}%
7786   \else
7787     \addto@hook\markdownLaTeXTable{#1\\}%
7788     \expandafter\@gobble
7789   \fi\markdownLaTeXRenderTableCell}
7790 \fi

```

**3.3.4.6 YAML Metadata** The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

7791 \ExplSyntaxOn
7792 \keys_define:nn
7793 { markdown/jekyllData }
7794 {
7795   author .code:n = { \author{#1} },
7796   date   .code:n = { \date{#1}   },
7797   title  .code:n = { \title{#1}  },
7798 }

```

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.

```

7799 % TODO: Remove the command definition in TeX Live 2021.
7800 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
7801 \markdownSetup{
7802   rendererPrototypes = {
7803     jekyllDataEnd = {
7804 %       TODO: Remove the else branch in TeX Live 2021.
7805       \IfFormatAtLeastTF
7806         { 2020-10-01 }
7807         { \AddToHook{begindocument/end}{\maketitle} }
7808         {
7809           \ifx\@onlypreamble\@notprerr

```



```

7810         % We are in the document
7811         \maketitle
7812     \else
7813         % We are in the preamble
7814         \RequirePackage{etoolbox}
7815         \AfterEndPreamble{\maketitle}
7816     \fi
7817 }
7818 },
7819 },
7820 }
7821 \ExplSyntaxOff

```

### 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.

```

7822 \newcommand\markdownMakeOther{%
7823     \count0=128\relax
7824     \loop
7825         \catcode\count0=11\relax
7826         \advance\count0 by 1\relax
7827     \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.

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

```

7828 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
7829 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
7830 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
7831     \do\#\do\^\do\_do\%do\~}%
7832 \input markdown/markdown

```

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

```

7833 \def\markdownMakeOther{%
7834   \count0=128\relax
7835   \loop
7836     \catcode\count0=11\relax
7837     \advance\count0 by 1\relax
7838   \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.

```

7839   \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

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. [11, sec. 31]. According to Eijkhout [12, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)T<sub>E</sub>X, but ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```

7840 \ifx\startluacode\undefined % MkII
7841   \begingroup
7842     \catcode`\|=0%
7843     \catcode`\|=12%
7844     |gdef|startmarkdown{%
7845       |markdownReadAndConvert{\stopmarkdown}%
7846                                   {|\stopmarkdown}}%
7847     |gdef|stopmarkdown{%
7848       |markdownEnd}%
7849   |endgroup
7850 \else % MkIV
7851   \startluacode
7852     document.markdown_buffering = false
7853     local function preserve_trailing_spaces(line)
7854       if document.markdown_buffering then
7855         line = line:gsub("[ \t][ \t]$", "\t\t")
7856       end
7857       return line
7858     end
7859     resolvers.installinputlinehandler(preserve_trailing_spaces)
7860   \stopluacode
7861   \begingroup
7862     \catcode`\|=0%
7863     \catcode`\|=12%
7864   |gdef|startmarkdown{%

```

```

7865     |ctxlua{document.markdown_buffering = true}%
7866     |markdownReadAndConvert{\stopmarkdown}%
7867                                     {|\stopmarkdown}}%
7868     |gdef|stopmarkdown{%
7869     |ctxlua{document.markdown_buffering = false}%
7870     |markdownEnd}%
7871 |endgroup
7872 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

7873 \def\markdownRendererLineBreakPrototype{\blank}%
7874 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
7875 \def\markdownRendererRightBracePrototype{\textbraceright}%
7876 \def\markdownRendererDollarSignPrototype{\textdollar}%
7877 \def\markdownRendererPercentSignPrototype{\percent}%
7878 \def\markdownRendererUnderscorePrototype{\textunderscore}%
7879 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
7880 \def\markdownRendererBackslashPrototype{\textbackslash}%
7881 \def\markdownRendererTildePrototype{\textasciitilde}%
7882 \def\markdownRendererPipePrototype{\char`|}%
7883 \def\markdownRendererLinkPrototype#1#2#3#4{%
7884   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
7885   \fi}\tt<\hyphenatedurl{#3}>}}%
7886 \usemodule[database]
7887 \defineseparatedlist
7888   [MarkdownConTeXtCSV]
7889   [separator={,},
7890   before=\bTABLE,after=\eTABLE,
7891   first=\bTR,last=\eTR,
7892   left=\bTD,right=\eTD]
7893 \def\markdownConTeXtCSV{csv}
7894 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7895   \def\markdownConTeXtCSV@arg{#1}%
7896   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
7897     \placetable[] [tab:#1]{#4}{%
7898       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
7899   \else
7900     \markdownInput{#3}%
7901   \fi}%
7902 \def\markdownRendererImagePrototype#1#2#3#4{%
7903   \placefigure[] []{#4}{\externalfigure[#3]}%
7904 \def\markdownRendererULBeginPrototype{\startitemize}%
7905 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
7906 \def\markdownRendererULItemPrototype{\item}%
7907 \def\markdownRendererULEndPrototype{\stopitemize}%

```

```

7908 \def\markdownRendererU1EndTightPrototype{\stopitemize}%
7909 \def\markdownRendererO1BeginPrototype{\startitemize[n]}%
7910 \def\markdownRendererO1BeginTightPrototype{\startitemize[packed,n]}%
7911 \def\markdownRendererO1ItemPrototype{\item}%
7912 \def\markdownRendererO1ItemWithNumberPrototype#1{\sym{#1.}}%
7913 \def\markdownRendererO1EndPrototype{\stopitemize}%
7914 \def\markdownRendererO1EndTightPrototype{\stopitemize}%
7915 \definedescription
7916   [MarkdownConTeXtDlItemPrototype]
7917   [location=hanging,
7918     margin=standard,
7919     headstyle=bold]%
7920 \definestartstop
7921   [MarkdownConTeXtDlPrototype]
7922   [before=\blank,
7923     after=\blank]%
7924 \definestartstop
7925   [MarkdownConTeXtDlTightPrototype]
7926   [before=\blank\startpacked,
7927     after=\stoppacked\blank]%
7928 \def\markdownRendererDlBeginPrototype{%
7929   \startMarkdownConTeXtDlPrototype}%
7930 \def\markdownRendererDlBeginTightPrototype{%
7931   \startMarkdownConTeXtDlTightPrototype}%
7932 \def\markdownRendererDlItemPrototype#1{%
7933   \startMarkdownConTeXtDlItemPrototype{#1}}%
7934 \def\markdownRendererDlItemEndPrototype{%
7935   \stopMarkdownConTeXtDlItemPrototype}%
7936 \def\markdownRendererDlEndPrototype{%
7937   \stopMarkdownConTeXtDlPrototype}%
7938 \def\markdownRendererDlEndTightPrototype{%
7939   \stopMarkdownConTeXtDlTightPrototype}%
7940 \def\markdownRendererEmphasisPrototype#1{\em#1}%
7941 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
7942 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
7943 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
7944 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
7945 \def\markdownRendererInputFencedCodePrototype#1#2{%
7946   \ifx\relax#2\relax
7947     \typefile{#1}%
7948   \else

```

The code fence infostrng is used as a name from the ConT<sub>E</sub>Xt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

```

```

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

```

```

7949   \typefile[#2] []{#1}%
7950   \fi}%
7951   \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
7952   \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
7953   \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
7954   \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
7955   \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
7956   \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
7957   \def\markdownRendererHorizontalRulePrototype{%
7958     \blackrule[height=1pt, width=\hsize]}%
7959   \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
7960   \stopmodule\protect

```

There is a basic implementation of tables.

```

7961   \newcount\markdownConTeXtRowCounter
7962   \newcount\markdownConTeXtRowTotal
7963   \newcount\markdownConTeXtColumnCounter
7964   \newcount\markdownConTeXtColumnTotal
7965   \newtoks\markdownConTeXtTable
7966   \newtoks\markdownConTeXtTableFloat
7967   \def\markdownRendererTablePrototype#1#2#3{%
7968     \markdownConTeXtTable={}%
7969     \ifx\empty#1\empty
7970       \markdownConTeXtTableFloat={%
7971         \the\markdownConTeXtTable}%
7972     \else
7973       \markdownConTeXtTableFloat={%
7974         \placetable{#1}{\the\markdownConTeXtTable}}%
7975     \fi
7976   \begingroup
7977   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
7978   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
7979   \setupTABLE[r][1][topframe=on, bottomframe=on]
7980   \setupTABLE[r][#1][bottomframe=on]

```

```

7981 \markdownConTeXtRowCounter=0%
7982 \markdownConTeXtRowTotal=#2%
7983 \markdownConTeXtColumnTotal=#3%
7984 \markdownConTeXtRenderTableRow}
7985 \def\markdownConTeXtRenderTableRow#1{%
7986 \markdownConTeXtColumnCounter=0%
7987 \ifnum\markdownConTeXtRowCounter=0\relax
7988 \markdownConTeXtReadAlignments#1%
7989 \markdownConTeXtTable={\bTABLE}%
7990 \else
7991 \markdownConTeXtTable=\expandafter{%
7992 \the\markdownConTeXtTable\bTR}%
7993 \markdownConTeXtRenderTableCell#1%
7994 \markdownConTeXtTable=\expandafter{%
7995 \the\markdownConTeXtTable\eTR}%
7996 \fi
7997 \advance\markdownConTeXtRowCounter by 1\relax
7998 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
7999 \markdownConTeXtTable=\expandafter{%
8000 \the\markdownConTeXtTable\eTABLE}%
8001 \the\markdownConTeXtTableFloat
8002 \endgroup
8003 \expandafter\gobbleoneargument
8004 \fi\markdownConTeXtRenderTableRow}
8005 \def\markdownConTeXtReadAlignments#1{%
8006 \advance\markdownConTeXtColumnCounter by 1\relax
8007 \if#1d%
8008 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8009 \fi\if#1l%
8010 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8011 \fi\if#1c%
8012 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
8013 \fi\if#1r%
8014 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
8015 \fi
8016 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8017 \expandafter\gobbleoneargument
8018 \fi\markdownConTeXtReadAlignments}
8019 \def\markdownConTeXtRenderTableCell#1{%
8020 \advance\markdownConTeXtColumnCounter by 1\relax
8021 \markdownConTeXtTable=\expandafter{%
8022 \the\markdownConTeXtTable\bTD#1\eTD}%
8023 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8024 \expandafter\gobbleoneargument
8025 \fi\markdownConTeXtRenderTableCell}
8026 \def\markdownRendererTickedBox{${\boxtimes$}}
8027 \def\markdownRendererHalfTickedBox{${\boxdot$}}

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Feb. 2017. URL: <http://www.luatex.org/svn/trunk/manual/luatex.pdf> (visited on 01/08/2018).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [5] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [6] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [7] Vít Novotný. *L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).
- [8] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [9] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [10] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [11] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [12] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

## Index

\author

232

<code>\autocites</code>	224
<code>blankBeforeBlockquote</code>	13
<code>blankBeforeCodeFence</code>	14
<code>blankBeforeHeading</code>	14
<code>breakableBlockquotes</code>	14
<code>cacheDir</code>	13, 17, 36, 37, 174
<code>citationNbsps</code>	15
<code>citations</code>	15, 64, 65
<code>\cite</code>	224
<code>\citep</code>	224
<code>\citet</code>	224
<code>codeSpans</code>	16
<code>compactdesc</code>	4
<code>compactenum</code>	4
<code>compactitem</code>	4
<code>contentBlocks</code>	16
<code>contentBlocksLanguageMap</code>	17, 179
<code>convert</code>	205
<code>\csvautotabular</code>	4
<code>\date</code>	232
<code>defaultOptions</code>	7, 32, 195
<code>\definetyping</code>	236
<code>definitionLists</code>	17, 54
<code>\directlua</code>	4, 37, 212
<code>eagerCache</code>	17, 18
<code>\enableregime</code>	233
<code>\endmarkdown</code>	73
<code>entities.char_entity</code>	136
<code>entities.dec_entity</code>	136
<code>entities.hex_entity</code>	136
<code>escape_citation</code>	175, 176
<code>escaped_citation_chars</code>	175, 175
<code>expandtabs</code>	159
<code>expectJekyllData</code>	18
<code>extension</code>	88
<code>extensions</code>	175
<code>extensions.citations</code>	175
<code>extensions.content_blocks</code>	179
<code>extensions.definition_lists</code>	182



extensions.fenced_code	184
extensions.footnotes	186
extensions.header_attributes	188
extensions.jekyll_data	189
extensions.pipe_table	192
fencedCode	19, 58, 215
\filecontents	206
finalizeCache	13, 17, 18, 20, 21, 36, 174
footnotes	20, 64
frozenCacheCounter	21, 174, 213
frozenCacheFileName	13, 20, 174
\g_luabridge_error_output_filename_str	38
\g_luabridge_helper_script_filename_str	37
\g_luabridge_output_dirname_str	38
hardLineBreaks	21
hashEnumerators	22
headerAttributes	22, 26, 67, 68
html	23, 66, 67, 223
hybrid	23, 29, 30, 39, 44, 80, 139, 160, 212
\includegraphics	4
inlineFootnotes	23
\input	34, 87, 208, 211
isdir	3
iterlines	159
jekyllData	3, 18, 24, 58–61
\jobname	37, 38
\label	221
languages_json	179, 179
\maketitle	232
\markdown	73
markdown	73, 73, 214
markdown*	73, 73, 74, 214
\markdown_jekyll_data_concatenate_address:NN	201
\markdown_jekyll_data_pop:	202
\markdown_jekyll_data_push:nN	202
\markdown_jekyll_data_push_address_segment:n	200
\markdown_jekyll_data_set_keyval:Nn	202

<code>\markdown_jekyll_data_set_keyvals:nn</code>	202
<code>\markdown_jekyll_data_update_address_tls:</code>	201
<code>\markdownBegin</code>	34, 34, 35, 71, 73, 87
<code>\markdownEnd</code>	34, 34, 35, 71, 73, 87
<code>\markdownError</code>	70, 70
<code>\markdownExecute</code>	210
<code>\markdownExecuteDirect</code>	210, 210
<code>\markdownExecuteShellEscape</code>	209, 210
<code>\markdownIfOption</code>	205
<code>\markdownIfSnippetExists</code>	75
<code>\markdownInfo</code>	70
<code>\markdownInput</code>	34, 35, 73, 74, 212, 214
<code>\markdownInputFileStream</code>	206
<code>\markdownInputPlainTeX</code>	214
<code>\markdownLuaExecute</code>	209, 210, 211, 212
<code>\markdownLuaOptions</code>	204, 205
<code>\markdownLuaRegisterIBCallback</code>	72
<code>\markdownLuaUnregisterIBCallback</code>	72
<code>\markdownMakeOther</code>	70, 233
<code>\markdownMode</code>	4, 37, 38, 71, 71, 209, 211
<code>\markdownOptionCacheDir</code>	3, 83, 205, 217
<code>\markdownOptionErrorTempFileName</code>	38, 38, 211
<code>\markdownOptionFinalizeCache</code>	36, 36, 83
<code>\markdownOptionFrozenCache</code>	13, 20, 36, 37, 79, 80, 83, 216, 217
<code>\markdownOptionFrozenCacheFileName</code>	36
<code>\markdownOptionHelperScriptFileName</code>	37, 37–39, 210, 211
<code>\markdownOptionHybrid</code>	39, 83
<code>\markdownOptionInputTempFileName</code>	37, 207, 208
<code>\markdownOptionOutputDir</code>	38, 38
<code>\markdownOptionOutputTempFileName</code>	37, 38, 211
<code>\markdownOptionSmartEllipses</code>	83
<code>\markdownOptionStripPercentSigns</code>	40, 206, 207
<code>\markdownOutputFileStream</code>	206
<code>\markdownPrepare</code>	205
<code>\markdownPrepareLuaOptions</code>	204
<code>\markdownReadAndConvert</code>	71, 206, 214, 234
<code>\markdownReadAndConvertProcessLine</code>	207, 208
<code>\markdownReadAndConvertStripPercentSigns</code>	207
<code>\markdownReadAndConvertTab</code>	206
<code>\markdownRendererAttributeClassName</code>	67
<code>\markdownRendererAttributeIdentifier</code>	67
<code>\markdownRendererAttributeKeyValue</code>	67

<code>\markdownRendererBlockHtmlCommentBegin</code>	66
<code>\markdownRendererBlockHtmlCommentEnd</code>	66
<code>\markdownRendererBlockQuoteBegin</code>	57
<code>\markdownRendererBlockQuoteEnd</code>	57
<code>\markdownRendererCite</code>	64, 65
<code>\markdownRendererCodeSpan</code>	47
<code>\markdownRendererCodeSpanPrototype</code>	86
<code>\markdownRendererContentBlock</code>	48, 48
<code>\markdownRendererContentBlockCode</code>	49
<code>\markdownRendererContentBlockOnlineImage</code>	48
<code>\markdownRendererDlBegin</code>	54
<code>\markdownRendererDlBeginTight</code>	54
<code>\markdownRendererDlDefinitionBegin</code>	55
<code>\markdownRendererDlDefinitionEnd</code>	55
<code>\markdownRendererDlEnd</code>	55
<code>\markdownRendererDlEndTight</code>	56
<code>\markdownRendererDlItem</code>	54
<code>\markdownRendererDlItemEnd</code>	55
<code>\markdownRendererDocumentBegin</code>	42
<code>\markdownRendererDocumentEnd</code>	42
<code>\markdownRendererEllipsis</code>	26, 44
<code>\markdownRendererEmphasis</code>	56, 85
<code>\markdownRendererFootnote</code>	64
<code>\markdownRendererHalfTickedBox</code>	41
<code>\markdownRendererHeaderAttributeContextBegin</code>	68
<code>\markdownRendererHeaderAttributeContextEnd</code>	68
<code>\markdownRendererHeadingFive</code>	63
<code>\markdownRendererHeadingFour</code>	63
<code>\markdownRendererHeadingOne</code>	62
<code>\markdownRendererHeadingSix</code>	63
<code>\markdownRendererHeadingThree</code>	62
<code>\markdownRendererHeadingTwo</code>	62
<code>\markdownRendererHorizontalRule</code>	64
<code>\markdownRendererImage</code>	48
<code>\markdownRendererImagePrototype</code>	86
<code>\markdownRendererInlineHtmlComment</code>	66
<code>\markdownRendererInlineHtmlTag</code>	66
<code>\markdownRendererInputBlockHtmlElement</code>	67
<code>\markdownRendererInputFencedCode</code>	58
<code>\markdownRendererInputVerbatim</code>	57
<code>\markdownRendererInterblockSeparator</code>	43
<code>\markdownRendererJekyllDataBegin</code>	58

<code>\markdownRendererJekyllDataBoolean</code>	60
<code>\markdownRendererJekyllDataEmpty</code>	61
<code>\markdownRendererJekyllDataEnd</code>	58
<code>\markdownRendererJekyllDataMappingBegin</code>	59
<code>\markdownRendererJekyllDataMappingEnd</code>	59
<code>\markdownRendererJekyllDataNumber</code>	61
<code>\markdownRendererJekyllDataSequenceBegin</code>	59
<code>\markdownRendererJekyllDataSequenceEnd</code>	60
<code>\markdownRendererJekyllDataString</code>	61
<code>\markdownRendererLineBreak</code>	43
<code>\markdownRendererLink</code>	47, 85
<code>\markdownRendererNbsp</code>	44
<code>\markdownRendererOlBegin</code>	51
<code>\markdownRendererOlBeginTight</code>	52
<code>\markdownRendererOlEnd</code>	53
<code>\markdownRendererOlEndTight</code>	53
<code>\markdownRendererOlItem</code>	27, 52
<code>\markdownRendererOlItemEnd</code>	52
<code>\markdownRendererOlItemWithNumber</code>	27, 52
<code>\markdownRendererStrongEmphasis</code>	56
<code>\markdownRendererTable</code>	65
<code>\markdownRendererTextCite</code>	65
<code>\markdownRendererTickedBox</code>	41
<code>\markdownRendererUlBegin</code>	49
<code>\markdownRendererUlBeginTight</code>	50
<code>\markdownRendererUlEnd</code>	51
<code>\markdownRendererUlEndTight</code>	51
<code>\markdownRendererUlItem</code>	50
<code>\markdownRendererUlItemEnd</code>	50
<code>\markdownRendererUntickedBox</code>	41
<code>\markdownSetup</code>	74, 74, 214, 218
<code>\markdownSetupSnippet</code>	75, 75
<code>\markdownWarning</code>	70
<code>new</code>	7, 195
<code>os.execute</code>	71, 210
<code>\PackageError</code>	73, 214
<code>\PackageInfo</code>	73, 214
<code>\PackageWarning</code>	73, 214
<code>parsers</code>	146, 158
<code>parsers.commented_line</code>	148

<code>\pdfshellescape</code>	209
<code>pipeTables</code>	6, 24, 28, 65
<code>preserveTabs</code>	25, 27, 159
<code>print</code>	210, 212
<code>reader</code>	88, 88, 146, 158, 175
<code>reader-&gt;convert</code>	174, 195
<code>reader-&gt;create_parser</code>	159
<code>reader-&gt;normalize_tag</code>	159
<code>reader-&gt;parser_functions</code>	159
<code>reader-&gt;parser_functions.name</code>	159
<code>reader-&gt;parsers</code>	158, 158
<code>reader-&gt;syntax</code>	170
<code>reader-&gt;writer</code>	158
<code>reader.new</code>	158, 158
<code>relativeReferences</code>	25
<code>\shellescape</code>	209
<code>shiftHeadings</code>	6, 26
<code>slice</code>	6, 22, 26, 137
<code>smartEllipses</code>	26, 44
<code>\startmarkdown</code>	87, 87, 234
<code>startNumber</code>	27, 52
<code>status.shell_escape</code>	209
<code>\stopmarkdown</code>	87, 87, 234
<code>stripIndent</code>	27, 159
<code>tableCaptions</code>	6, 28
<code>taskLists</code>	28, 41, 222
<code>tex.print</code>	210, 212
<code>texComments</code>	29, 160
<code>\textcites</code>	224
<code>tightLists</code>	29, 50–54, 56, 218
<code>\title</code>	232
<code>underscores</code>	30
<code>\url</code>	4
<code>\usepackage</code>	73, 76
<code>\usetheme</code>	76
<code>util.cache</code>	88
<code>util.err</code>	88
<code>util.escaper</code>	91
<code>util.expand_tabs_in_line</code>	89

util.flatten	90
util.intersperse	90
util.map	91
util.pathname	92
util.rope_last	90
util.rope_to_string	90
util.table_copy	89
util.walk	89, 90
\VerbatimInput	4
writer	88, 88, 136, 175
writer->active_attributes	143
writer->active_headings	143
writer->block_html_comment	141
writer->block_html_element	142
writer->blockquote	142
writer->bulletlist	140
writer->cacheDir	137
writer->citation	176
writer->citations	176
writer->code	139
writer->codeFence	184
writer->contentblock	180
writer->defer_call	146, 146
writer->definitionlist	182
writer->document	143
writer->ellipsis	138
writer->emphasis	142
writer->escape	139, 139
writer->escape_minimal	139, 139, 176
writer->escape_uri	139, 139
writer->escaped_chars	139, 139
writer->escaped_minimal_strings	138, 139
writer->escaped_uri_chars	138, 139
writer->get_state	146
writer->heading	143
writer->hrule	138
writer->hybrid	137, 176
writer->image	140
writer->inline_html_comment	141
writer->inline_html_tag	141

writer->interblocksep	138
writer->is_writing	137, 137
writer->jekyllData	189
writer->linebreak	138
writer->link	139
writer->nbsp	137
writer->note	186
writer->ollist	140
writer->pack	138, 174
writer->paragraph	138
writer->plain	137
writer->set_state	146
writer->slice_begin	137
writer->slice_end	137
writer->space	137
writer->string	139
writer->strong	142
writer->suffix	137
writer->table	193
writer->checkbox	142
writer->uri	139
writer->verbatim	142
writer.new	136, 136
\writestatus	233